

A Common Person's WordPerfect Macro Manual

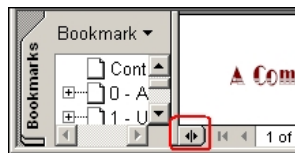
© 2004 by Doug Loudenback, Oklahoma City, OK, All Rights Reserved

RELEASE NOTES

[Click here to move to Contents](#)

[Click here to move to the Index](#)

Hyperlinks in this document are [blue like this \(for external web/email links\)](#) or [red like this \(for links within this document\)](#), are "bold," and are NOT underscored.



In Adobe Acrobat Reader, this document initially opens without Bookmarks showing, even though it is fully bookmarked. To turn off/on Bookmarks, click the Bookmarks "Tab" in Adobe Acrobat Reader or the open/close button highlighted here. If a topic has sub-bookmarks, it has a "+" sign to its left. Click the + sign to expand the "tree" for a particular bookmark. Use other features of Adobe Acrobat Reader to select, print, copy, search, etc., as are needed.

PREFACE

A better title than "Preface" is, "Limitations of the Skill Level of This Writer." Although my skills are good enough to write a commercially successful macro program for Oklahoma lawyers, *I am by no means a world-class macro writer*. I've had to grope for everything I've learned and I inevitably learn better "macro ways" than I knew before. Mostly, I am a pragmatist who learns on an "as needed" basis. In the process of writing of this manual I've learned lots of better ways. [†]

So, why do I presume to write this manual? (1) Not a lot of literature is readily available to assist "common persons" like me. (2) I want to give other "common persons" some good news – a common person can write outstandingly useful macros, even if a real macro guru would do it differently. *So, get out your "book" and write this down: You don't need to be a macro guru ... if a macro works, it works.* (3) I know that what I express here "works" since I've successfully done everything that is discussed here works as described and nothing I state here is hypothetical, even if what I say could be done differently or better than I have done in these descriptions.

This manual embraces macro topics in WordPerfect 6.1 through 12.0, but it's probably better for WordPerfect 8.0 and higher since I no longer use WordPerfect 6.1 or 7.0. I understand that macros for WordPerfect 6.0 and 6.1 for Windows were very similar, but I have no personal knowledge of that.

This manual doesn't contain all that *I think* I've learned, and, obviously, it contains nothing about that which I don't yet know. Most importantly, I hope that it contains nothing that will lead you astray because of my own ignorance. Be sure to see Chapter 1, [Additional Resources](#). The [Table of Contents](#) is immediately below.

This manual has two intended audiences: (1) people who already have moderate-to-better skill levels; and (2) people who don't and have not yet dipped their toes into macro waters at all. To all, I wish you a Good Macro Journey!

- **MODERATE-TO-ADVANCED USERS.** For you, or for beginners who want to be "you", you don't need the pep-talk. You just need reference information to help you solve particular kinds of tasks you're working on – these links take you to the [Index](#), [Glossary](#), [Math](#), [Date](#), [Dialog Show and Callback Routines](#), or [Macro Examples](#) Chapters.
- **NON-MACRO USERS.** You cannot begin to know what powerful WordPerfect tools you miss by not using macros routinely, day-to-day. But, many WordPerfect users are intimidated by even the thought of learning to write macros – either that or they just don't see the need. *Well, that's all garbage! You don't have to be a macro guru* to write exceptionally useful WordPerfect macros which make your work faster and more pleasurable. It's not like you have to learn a *whole* foreign language – you don't need to be fluent. You can write a perfectly usable macro today, in just a few minutes. Learn more later when the need or desire exists. I never had a macro writing class – and for any real macro gurus who may read this, I know that it shows! But, being self-taught, I'm here to say that if I can do it, you can do it, too. If you are just beginning, you may want to start with [Understanding What Macros Are...](#) and/or [First Things](#).

[†] Thanks are given to J. Dan Broadhead, Barry MacDonnell, and Roy Lewis for their assistance in critiquing the initial draft of this manual, to Ken Hobson who has tried to teach me many things that I still don't quite grasp, and to many others for their encouragement! All were very helpful and I am grateful to each of them. My VERY special thanks is given to J. Dan Broadhead whose detailed critiques of this manual's drafts have improved it *very substantially* from what it would have been and, not the least, for his permission to use several of his comments herein (they are almost always "boxed"). That said, all errors in this paper are mine, none are theirs, and none of their respective contributions implies endorsement by any of them of this manual or any of the dumb things I may have said herein!

CONTENTS

NAVIGATION: On each page after this page, the [Manual Name](#) at the top of each page returns here. Each [Chapter Title](#) is a link to the NEXT CHAPTER. Within a chapter, [topics](#) and [subtopics](#) links are to the top of that chapter. [Dark Red links](#) are to locations in this manual. [Blue Links](#) are to Internet locations. [CommandName](#) links in chapters are usually links to that Command's description in Chapter 10, Glossary.

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

DETAILED CONTENTS

Chapter 1 – Additional Resources

- ! [WordPerfect Macro Help](#)
- ! [Limitations of the Help Files](#)
- ! [PerfectScript Command Browser](#)
- ! [Books/Manuals](#)
- ! [Useful Internet Downloads](#)
- ! [On-Line Q&A Forums](#)

Chapter 2 – Understanding What Macros Are, Where They Are, and How They Work

- ! [General Definition](#)
- ! [Storage Location of Your Macros](#)
- ! [Usual Macro File Location](#)
- ! [Shipping Macros](#)
- ! [How Macros Work](#)
- ! [Playing Macros](#)

Chapter 3 – First Things About Writing WordPerfect Macros

- ! [Wp10's Select Text Method Change](#)
- ! [Wp11's 11.0.0.225 & 11.0.0.300 Problem](#)
- ! [Writing Macros Using the Keyboard](#)
 - [Starting & Stopping the Recording](#)
 - [Naming the Macro](#)
 - [Using Shortcut Names](#)
 - [A Dumb Keyboard Macro Example](#)
- ! [Writing Macros From Scratch](#)

Chapter 4 – Controlling Macro Flow

- ! [Quit](#)
- ! [Go](#)
- ! [Call](#)
- ! [Return](#)
- ! [Label](#)
- ! [OnError](#)
- ! [OnNotFound](#)
- ! [OnCancel](#)

! [An Example](#)

Chapter 5 – Using the Dialog Editor During Macro Editing

- ! [Opening a Macro to Edit](#)
- ! [The Editing Environment](#)
- ! [The Macro Toolbar](#)
 - [The "Little" Buttons](#)
 - [The "Commands..." Button](#)
 - [The "Codes" Button](#)
 - [The "Dialog Editor" Button](#)
 - [Dialog Properties](#)
 - [Dialog Font](#)
 - [Dialog Editor Controls](#)
 - [Adding Controls](#)
 - [Control Names](#)
 - [Size and Placement](#)
 - [Associated Variable](#)
 - [Bitmap Control](#)
 - [Push Button Control](#)
 - [Radio Button Control](#)
 - [Check Box Control](#)
 - [Combo Box Control](#)
 - [Date Box Control](#)
 - [Edit Box Control](#)
 - [List Box Control](#)
 - [Static Text Box Control](#)
 - [File Viewer Control](#)
- [Aligning Stuff In The Dialog](#)
- [Ordering The Controls](#)
 - [Controlling The Sequence](#)
 - [Default Button](#)
 - [Initial Focus](#)
 - [When You're Done](#)
- ! [Enter Macro Code As Needed](#)

- ! Click the Save & Compile Button
- ! Errors & Warnings During Compilation

Chapter 6 – Math Routines

- ! Math Operators
- ! General Rules
- ! Automatic Data Conversion
 - Concatenation & Reduction
 - Forcing Presumption
 - Shifting Presumptions
 - Examples
- ! Addition
- ! Subtraction
- ! Multiplication
- ! Division
- ! The Floating Cell Problem
- ! Massaging Numbers
 - Working With Integers
 - Working With Decimals
 - NumStr
 - StrNum
 - StrPos
 - SubStr
 - StrLen
- ! Specific Tasks
 - Getting Rid of Commas
 - Making It Look Right
 - Adding Commas
 - Percentages
- ! Error Routines

Chapter 7 – Date Routines

- ! Date System Variables
- ! Common Date Commands In Documents
- ! Commands Not Covered
- ! Massaging Dates
- ! Date Error Trapping

Chapter 8 – DialogShow/Callback Routines

- ! Trouble in Common Person's River City
- ! DialogShow, Generally
 - Syntax
 - Using DialogShow Without A Callback
 - Use of DialogDismiss & DialogDestroy
- ! Why Callbacks?

- ! Non-Covered Callback Commands
- ! Callback Basics

- General Definition and Callback Elements
- One Callback At A Time
- Region Names, Generally
- RegionGet... Commands
 - RegionGetCheck
 - RegionGetSelectedText
 - RegionGetWindowText
 - Putting RegionGet... Commands Together
 - Sample RegionGet... Code
- Other Region... Commands
 - RegionShowWindow
 - RegionResetList
 - RegionAddListItem
 - RegionRemoveListItem
 - RegionSelectListItem
 - RegionSetCheck
 - RegionSetFocus
 - RegionSetWindowText
- Making A Double-Click End A Callback
- WordPerfect Version Control
- Commands At Beginning Of DialogShow
- ! Learning By Example
 - First Example (Simplest)
 - Second Example (1st Example Expanded)
 - Third Example (Excerpts From Math.wcm)
 - Commands Within The Callback Label
 - Using DialogShow During Callbacks
 - Fourth Example (WaitMessage)

Chapter 9 – Macro Examples

- ! Math.wcm
- ! ConvertFE.wcm
- ! Wp9select.wcm

Chapter 10 – Glossary of Macro Commands

- ! General Notes
- ! Math and Comparison Characters
- ! System Variables
- ! All The Rest

Index – Hyperlinked Manual Index

Release Notes

NOTES

Chapter 1

ADDITIONAL RESOURCES

Links: The [Chapter Name](#), above, moves to the next chapter. 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.

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				
WordPerfect Macro Help		Limitations of On-Line Help		PerfectScript Command Browser
Books/Manuals		Useful Internet Downloads		On-Line Q&A Forums

WordPerfect's on-line macro help, and various quality resources are listed here. Internet links sometime change. At the time of this writing, the Internet links below are current.

! WORDPERFECT'S MACRO HELP. In many ways, the built-into WordPerfect Macros Help file is really quite good ... but in other ways, it's got to be judged as inadequate and incomplete. Nonetheless, your first attempt to find help should be WordPerfect's Macros Help since it's immediately available and may very well give you the answer you are looking for.

- **Where ARE The Help Files?** Unless you did a "custom" install and made sure that the Macros Help file was included in the installation, the main Macros Help file probably did not install automatically in all of the various WordPerfect versions. To "know" if you have installed the help file, using Start, Find, Files or Folders, search for the file matching the version of WordPerfect you are running, as shown below. If you don't find the main macros help file, you will need to do a supplemental installation from your original Disk 1 Installation CD.

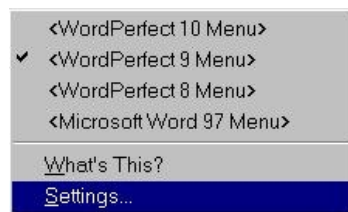
WordPerfect 6.1	Main File: The file is wpmh61us.hlp. Unless you changed the default location during install, the file is probably in the C:\Office\WpWin directory. As I no longer have access to Wp6.1, PerfectScript help files similar to those for identified for WordPerfect 8.0 and higher may also be available, I don't know.
WordPerfect 7.0	Main file: wpmh7us.hlp. Unless you changed the default location during install, the file is probably in the C:\Corel\Office7\Shared\Help7 directory. As I no longer have access to Wp7.0, PerfectScript help files similar to those identified for WordPerfect 8.0 and higher may also be available, I don't know.
WordPerfect 8.0	Main file: wpmh8en.hlp. Two others are psmh80en.hlp (PerfectScript macros help) and ps80en.hlp (PerfectScript Utility Help). Unless you changed the default location during install, the files are probably in the C:\Corel\Suite8\Shared\Help directory.
WordPerfect 9.0	Main file: wpmh9en.hlp. Others are psmh90en.hlp (PerfectScript macros help) and ps90en.hlp (PerfectScript Utility Help). Unless you changed the default location during install, the files are probably in the C:\Program Files\Corel\WordPerfect Office 2000\Shared\Help directory.
WordPerfect 10.0	Main file: wpmh10en.hlp. Others are psmh10en.hlp (PerfectScript Macros help), and ps10en.hlp (PerfectScript Utility help). Unless you changed the default location during install, the files are probably in the C:\Program Files\Corel\WordPerfect Office 2002\Shared\Help directory.

WordPerfect 11.0	Main file: wpmh11en.hlp. Others are psmh11en.hlp (PerfectScript Macros help) and ps11en.hlp (PerfectScript Utility help). Unless you changed the default location during install, the files are probably in C:\Program Files\WordPerfect Office 11\Shared\Help directory.
WordPerfect 12.0	Main file: wpmh12en.hlp. Others are psmh12en.hlp (PerfectScript Macros help) and ps12en.hlp (PerfectScript Utility help). Unless you changed the default location during install, the files are probably in C:\Program Files\WordPerfect Office 12\Shared\Help directory.

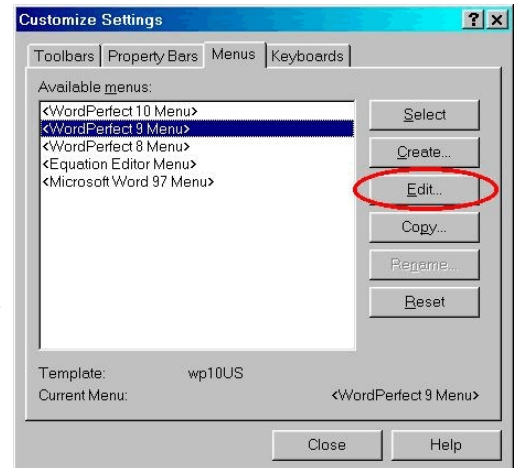
Assuming the appropriate Macros Help files are installed by you in the 1st place, it would be nice if you could just select them in some menu within WordPerfect, such as in the Help Menu. But, it's not always that simple.

Regrettably, for some WordPerfect versions, if you want the main WordPerfect Macros Help to be immediately accessible from your Help and/or Tools | Macros... Menu(s), you will need to edit your Menu to have instant access to that help file. These images show you how to add the main WordPerfect Macro help file to your Help menu list. The images shown here are WordPerfect 10 but the process is similar for other versions.

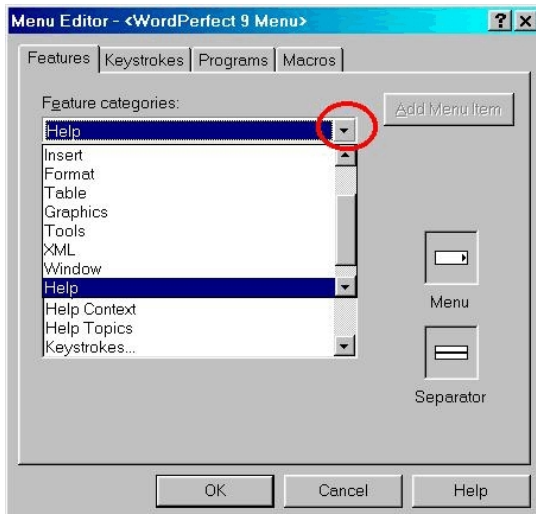
To add the main on-line Macros Help file:



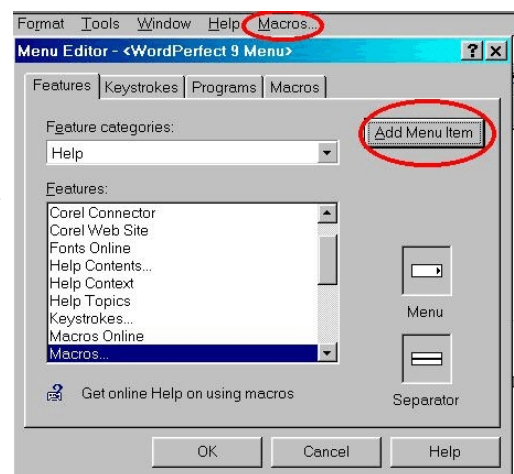
1st, right-click anywhere on the Menu (it is the line containing the words such as, File, Edit, View, Insert, Format, Tools, Window, Help). A popup menu will appear, and the Menu you are using will have a checkmark by it. In this menu, select Settings ...



2nd, having done so, the "Customize Settings" dialog will appear (right). If the "Menus" tab is not preselected, click on the



"Menus" tab. The dialog shown at the right will appear and the "Menu" you are editing will be selected. Click the "Edit" button in the right column containing the buttons. The Menu Editor dialog will open (left) and the dialog title bar will indicate the particular menu you have selected for editing will be in the name of the dialog title bar.



In the "Feature categories" box (right), click the ▼ at the right end of the box. From the drop-down menu which will appear, scroll down the list and select Help, as shown here. Having selected "Help" as the Feature category, scroll down the "Features" list until you find Macros... That item is the main Macros Help file.

Click the Add Menu Item button and that will add the Macros... item (this is the Macros help file) to your menu, as shown above.

You could click OK and stop at this point. But, probably, you don't want to leave your Menu as it is. It would be better, for example, to move the Macros... item to your regular Help Menu.

With the Menu Editor open, you can "drag" the Macros... item to the regular Help menu, or to the Tools, Macros... menu, or wherever you want, as is shown in this pair of images.

The top image shows the location of Macros... (the main macro help file) after the operation, above. Move your mouse pointer to the Macros... item and hold down the left mouse button on the item. Continuing to hold it down, drag Macros... over to the Help Menu and move down the Help Menu list. Here, I've stopped just below the regular Help Topics item. Release the mouse pointer. Now, click OK in the Menu Editor, and then Close in the Customize Settings dialog.

This done, now you can click Help, Macros... and the main Macros Help file should immediately appear.

Use the main Macros Help file just as you would any Windows help file. Click on a "book" to expand it. Click the "Index" tab for a chronological list of help items. Or, click the "Find" tab to expand the search capability, if you want. Note the "Print" button. You can print selections of the Help file to paper, if you want.

Doing the above will probably give you access to the PerfectScript Macros Help File (psmhXen.hlp), where "X" is 8, 9, 10, 11, or 12, since it is likely integrated with the main macros help file. But, that is probably not so concerning the PerfectScript Utility help file, psXen.hlp, where "X" is 8, 9, 10, 11, or 12. That file is typically only accessible AFTER opening the PerfectScript Utility from your Windows Start Menu. The file is NOT always in the list of Help files in Menu | Help "Help" files list. If not, you can't select that file from there.

But, you could make it accessible on your Menu bar by using the "Programs" tab instead of the "Features" tab in the Menu Editor, then selecting the Add Programs button, and then selecting "All Files" in the File Type box, and then fishing to directory and file psXen.hlp, where "X" is 8, 9, 10, 11, or 12. If you did that, you could then drag the file to the desired menu, e.g., the Help Menu item. That process is not illustrated in this manual.

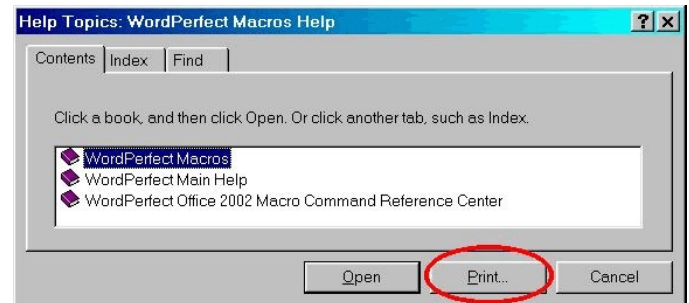
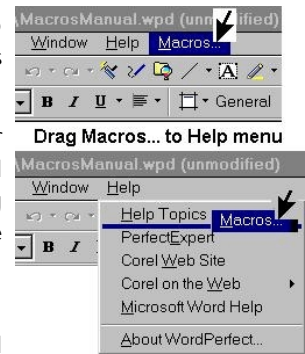
Why would you want to open this particular help file, regardless of how you do it? The psXen.hlp file contains information which is not included in the main on-line help file. If you're looking for answers, you want every available resource that you can easily get.

- **Limitations of the Help Files.** The Macros Help files, particularly the main on-line macros help files, have not been kept "up to date in Kansas City" as the various WordPerfect versions have been released and updated. While "new" macro commands are documented fairly well, "old" or "obsolete" commands have sometimes not been, and improved features in existing macro commands are sometimes not described at all.

Until Service Pack 4 of WordPerfect 9 (9.0.0.883) was released, macro compilation was not set up to alert you to "obsolete" but usually still valid WordPerfect macro commands. But, with Wp9's Service Pack 4, carried forward to WordPerfect 12, the macro compilation process now "warns" you if macro commands are "obsolete."

These warnings are "good" since, with such warnings, you are at least now "on notice" that you're using archaic macro language. They are "bad" because, when such warnings first show up, you'll be pulling out your hair wondering, "Huh? I've been doing XYZ for years now, without any problems at all, and I never heard anything like such a warning before." Barry MacDonnell discusses the development at <http://home.earthlink.net/~wptoolbox/ImportantNotes.html>.

But, the associated problem with the on-line macros help file is that, as late as WordPerfect 12, some commands are listed as current macro commands in the help file but which, when compiled, will be "flagged" with an obsolete warning message.



An example: *QuickCorrectQuickBulletsQry* and *QuickCorrectQuickIndentQry* were designed to return a Boolean True/False result if the particular WordPerfect features were turned on or off. If the "query" produced a *True* or *False* result, you could turn on or off the respective QuickCorrect... commands during the macro's execution, and, when that would be done, turn the QuickCorrect... commands on or off, as you would prefer. These macro commands are now deemed "obsolete" and produce warnings during the compilation of a macro in WordPerfect 9 (latest service pack) through 12 that contains them. Even so, the on-line WordPerfect Macro Help still includes a description of each command. A "common sense" user might think that these are valid, current commands, else they would not be described in WordPerfect's Macro Help file. Another WordPerfect 9.0.0.883 through Wp 12.0 example is *FileOpenDlg* ... it's described in the help file, but it is flagged as obsolete when the macro compiles.

At least in some instances, the on-line help fails to describe nifty features which have been added to existing macro commands. For example, J. Dan Broadhead, in his critiquing of a draft of this manual, noted that, beginning with WordPerfect 9.0.0.883 (Service Pack 4), it became possible to have more than one dialog using a callback routine at the same time. Who woulda' known?

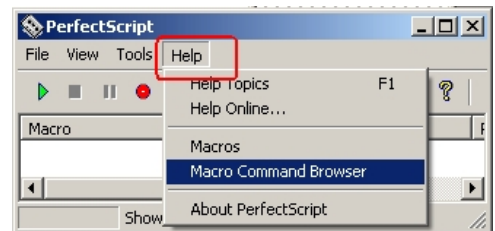
For more information about obsolete and/or improved routines, J. Dan Broadhead's *PerfectScript Ramblings* provides much additional insight: www.jdan.com/perfectscript/. There, select a link to WordPerfect x.x Changes (where x.x is the WordPerfect version number), and, with a file open, search (Ctrl+F in Windows Internet Explorer) for "Obsolete Commands and Language Features" and/or "Modified and Improved Commands and Language Features", or string subsets until you locate the section title.

Still, while the on-line Macro Help files must be regarded as flawed, even so, they are useful most of the time – I'd rather be with 'em than without 'em – but keep your eyes wide open.

- ! **PERFECTSCRIPT'S COMMAND BROWSER/INSERTER.** Sometimes unknown, often neglected, PerfectScript's Command Browser (aka Command Inserter) is a handy available resource for getting help. It can be accessed in one of two ways: (1) From the Commands... button in the Macro Toolbar; or (2) From the PerfectScript Utility. Either way, you get to the same place – although when you're editing a macro its use will be most convenient. Also, see Chapter 5's **Commands button** discussion.

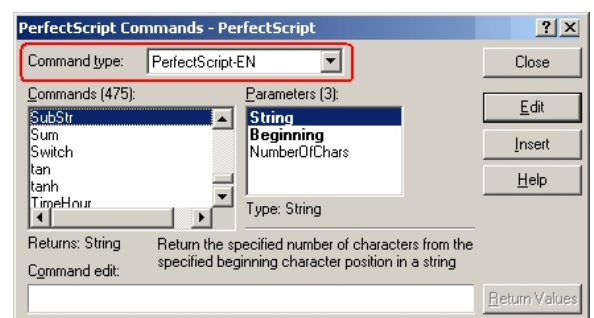
• Opening the Command Browser On Your Desktop.

Depending on your version of WordPerfect, to open the PerfectScript Utility use the Start | Programs, and find the program group for your WordPerfect Suite, e.g, WordPerfect Office 2000, WordPerfect Office 2002, WordPerfect Office 11, or WordPerfect Office 12. In that group, select Utilities | PerfectScript x (where "x" matches your version of WordPerfect) and the PerfectScript Utility will open. In its menu, select Help | Macro Command Browser.



• Opening the Command Browser In WordPerfect.

With a macro open for editing, click the Commands... button in the Macro Toolbar. For a picture of that Toolbar and button, see Chapter 5's **Macro Toolbar** and **Commands button** discussion. Clicking the Commands button opens the Command Browser aka Command Inserter.



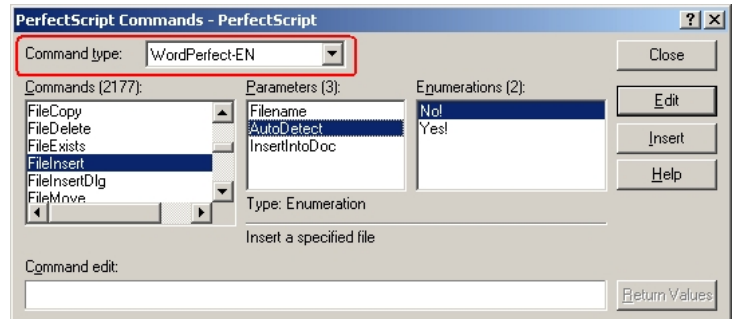
Command syntax is 100% reliable in the Command Browser, not true in WordPerfect's on-line macro help.

In the "Command type" box, you'll mainly be interested in 2 types - PerfectScript (in Wp10, shown here, PerfectScript-EN) or WordPerfect product commands, shown below. Locate a command (here, *SubStr* is shown) that you want information about and its parameters will appear in the Parameters box adjoining at the right. The selected parameter's "Type" appears below the Parameter box (here, the *String* parameter is shown to be a "String" Type, but if the *Beginning* parameter were selected, the message would read "Numeric"). A description of the command's use appears below that. To select a command in the Commands box, you can start typing in a command name or use the scroll bar to locate the command you want to know about. The above picture has been truncated here, but the entire dialog is shown in the next picture.

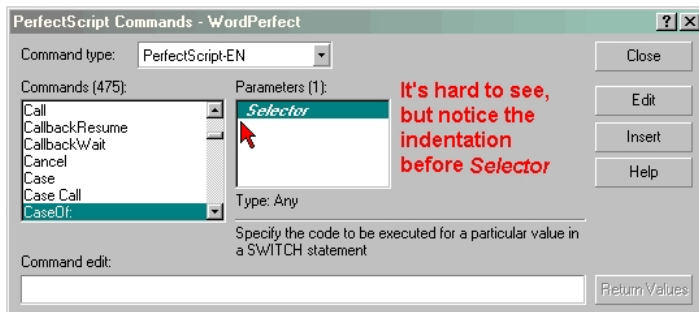
Here, in the Command Type box, WordPerfect (product commands) is selected and the *FileInsert* command is picked. The *AutoDetect* parameter is selected and, since Enumerations are possible for the *AutoDetect* parameter, they are shown in the Enumerations box, *No!* and *Yes!*.

Parameter and Enumeration boxes only appear IF a command uses either.

Here are some tips from J. Dan Broadhead (edited slightly by me) about using the Command Browser/Inserter – and if these tips are in WordPerfect's on-line macro help, I surely could not tell you where – I can't find them.

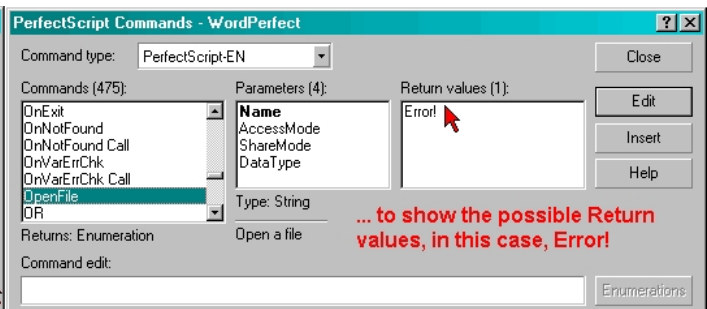
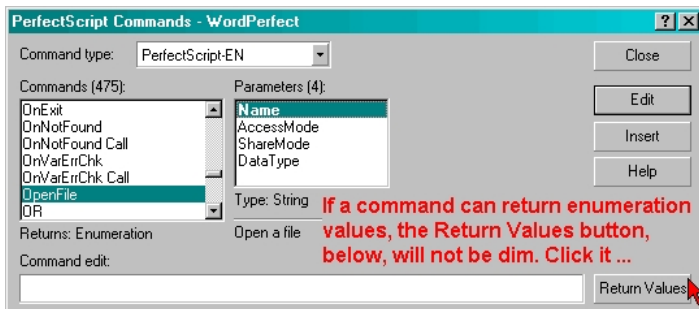


- (1) If a parameter name is bold, it is required; but, if not, it is optional.
- (2) If a parameter is indented and *italic*, then it can be repeated (e.g., Caseof:, shown below).
- (3) If you right-click on a command name, on-line help for that command immediately appears!
- (4) For commands can return enumeration values, the return enumeration values are not directly shown. Starting in WP9-SP4, there is a new button in the bottom right corner, called "Return Values". If a command returns an enumeration type, then this button is enabled, and clicking on it will change the "enumeration" list box from showing the enumerations for the parameters to showing the enumerations returned by the command. Try the OpenFile (shown below) or DefaultUnits commands in PerfectScript as examples.



Caseof: example. Notice the small space between Selector and the left margin of the Parameters box. That's an indent. Notice that Selector is (a) bold (required), (b) indented and italic (can be repeated, e.g., Caseof "OKBtn";1;"CancelBtn";2: will work just fine (eliminating the need for multiple Caseof: statements which will cause the same thing(s) to occur).

OpenFile example: Two pictures are shown below, a "before" and "after". Notice the changes.



Teaching point: Don't forget about using the Command Browser! It is a handy quick reference, and, unlike the on-line main macro help file, only "current" commands should appear. Also, see Chapter 5's **Commands** button discussion.

! **BOOKS/MANUALS.** If you're a "Common Person" when it comes to writing macros, you're going to need some help sooner or later beyond what is said in this manual and beyond the on-line Macro Help file. Here are some additional resources.

! **Core! Manuals.** The WordPerfect 8 Macro Manual is available at J. Dan Broadhead's website in html format, www.jdan.com/perfectscript/macros8.htm, and the WordPerfect 9 Macro Manual is

available at Texas A&M's website (in its biology department area, no less), www.bio.tamu.edu/help/manuals/corel2000/Macro9en.pdf or at Cornell Law School's intranet site, <http://support.law.cornell.edu/technology/downloads/pdf/macro9en.pdf>. (535 KB). Since Wp8 and higher macro commands are quite similar, these references may be useful to all those versions.

! **Gordon McComb's WordPerfect for Windows Macros, The Definitive Guide to Using Macros in WPWin** (about \$50), www.gmccomb.com/wp/. Many consider this book the "bible" of WordPerfect for Windows macros. While it expressly covers macro stuff in Wp6.1, 7.0 and 8.0, and even though some changes occurred in Wp9.0 and Wp10.0, for the most part Wp9.0 and higher macro commands and syntax are the same as in Wp8.0. Main shortcoming: the 450+ page, 8 ½ x 11 soft paper book is not indexed. This is the only WordPerfect for Windows macro book I've purchased and I recommend it.

! **Julie Jeppson's WordPerfect 9 Advanced Macro Programming: A Learning Guide** (\$70), www.wpmacros.com/books.htm, and other titles. This 8 ½ x 11 book is 700+ pages. The author is an expert on WordPerfect macros and other WordPerfect topics. Though I've not seen it, considering the source, the book is most probably worth the money for those interested in macro skills.

! **USEFUL INTERNET DOWNLOADS/INFORMATION.** The Internet contains lots of useful sites which provide help in writing WordPerfect macros. Some are these:

! **J. Dan Broadhead's PerfectScript Ramblings.** www.jdan.com/perfectscript. J. Dan Broadhead was one of the primary developers of PerfectScript since WordPerfect for Windows 6.0, and, with WordPerfect 9.0's release, he became Corel's sole developer of PerfectScript, ending with the initial release of WordPerfect 10.0, after which his affiliation with Corel ended. Since then, no one appears to be at home (so to speak) in Corel concerning PerfectScript development. JDan is a forum moderator and contributes his extraordinary knowledge at *WordPerfect Universe's User-To-User Forums*, www.wpuniverse.com. Several of his comments are contained in this manual and, perhaps most profoundly, in Chapter 8, **DialogShow & Callback Routines**.

! **Barry MacDonnell's Toolbox For WordPerfect For Windows.** <http://home.earthlink.net/~wptoolbox/Homepage.html>. This excellent website includes numerous macros, tips and templates for WordPerfect for Windows. As importantly, it includes links to downloadable macros from various Internet and other sources. But, of great importance if you are a WordPerfect 10 or later user, he describes some very important variances in macro operation between pre-and-post WordPerfect 10 versions which are related to a generic change WordPerfect's text selection method beginning in WordPerfect 10 see <http://home.earthlink.net/~wptoolbox/WP9TSS.html> – and he discusses these topics more than anywhere else I know on the Web. Be sure to visit his website. His discussion of the WordPerfect 10 issues is so important that I've taken the liberty in this paper of including a "BMT" link in various affected commands in **Chapter 10's "Commands/Routines"** table and in **Chapter 3's "First Things..."**

! **Julie Jeppson's PerfectScript Journal and other resources.** <http://www.wpmacros.com/psjournal/index.htm>. Ms. Jeppson's offerings include many articles available in her archives, and several macro "snippets" are downloadable. Ms. Jeppson, a former employee of WordPerfect Corporation and its successor, Novell, is a highly regarded source for many WordPerfect topics. Her credentials are here: <http://www.wpmacros.com/about.htm>.

! **Seth Katz's WordPerfect Macro Tutorial.** www.shkatz.com/macrotut/index.html. This is a fine html tutorial for those who are beginning to write WordPerfect macros, the intended audience.

! **Other Links at WordPerfect Universe.** www.wpuniverse.com/links.html. While WPU's highest value to me is its User-To-User forum, this link to its "links" page accesses numerous other helpful macro (and other WordPerfect) items of interest.

- **My Website's WordPerfect Area.** www.dougloudenback.com/wp.htm. Resources beyond this manual are there, perhaps the most useful being the **WordPerfect 10 Macro Commands** PDF document, a 121 page hypertexted document describing the 2175 System and Product Commands and the 473 PerfectScript commands for WordPerfect 10. These commands are substantially the same for WordPerfect 11.0 and 12.0.

! **ON-LINE Q & A FORUMS.** Some Internet locations provide interactive help. Two are these:

! **THE VERY BEST.** *WordPerfect Universe's User-To-User Forums*. www.wpuniverse.com. This question/answer Internet forum has proven absolutely invaluable to me in figuring out answers to the nastiest of WordPerfect macro issues (among the numerous other WordPerfect subject areas). World-class WordPerfect experts moderate the forums. Nothing more need be said.

! **Corel's WP Newsgroups.** http://support.corel.com/scripts/rightnow.cfg/php.exe/enduser/std_adp.php?p_faqid=754345. While "newsgroups" by nature can sometimes be notable for user "carping", still, if you can stand or avoid the griping that participants sometimes seem intent upon doing and just post your questions, knowledgeable C-Techs are quite helpful with answers within a few hours or so. While I much prefer *WordPerfect Universe's User-To-User Forums*, still, the newsgroups provide another option for obtaining help. I note that some avoid the newsgroups, precisely because of the griping that some newsgroup users seem to be intent on doing, and the C-Techs there are more tolerant of non-constructive speech than I am. WordPerfect's macro stuff isn't "perfect" but nothing else can hold a candle to it. If you are discerning and tolerant, the messages can be helpful.

[Top of Chapter 1](#) [Chapter 2](#)

NOTES

Chapter 2

Understanding What Macros Are,
Where They Are, and How They Work

Links: The [Chapter Name](#), above, moves to the next chapter. 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.

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 Definition	Storage Location of Macros	Usual Macro File Location	Shipping Macros	
How Macros Work		Playing Macros		

- **GENERAL DEFINITION.** In simplest terms, a WordPerfect macro is WordPerfect document written in WordPerfect's macro language which, when compiled, i.e., translated into an object file (a hidden area of the macro that you never see), it does something. The "something" varies from performing very simple WordPerfect tasks which you could do in WordPerfect to enormously complex tasks which you could probably not do in WordPerfect in any other way. An example of a simple task might be to automate a signature block for a letter or a court document. An example of an enormously complex task is the mathematical computation of Oklahoma's Child Support Guidelines law based upon numerical data you enter into a dialog. If you are an Oklahoma lawyer, you know that performing such a task involves a blend of each parent's income, number of minor children, day care expense (and who's paying it), health insurance expense (and who's paying it), what statutory year is involved with the computation, whether or not "shared parenting" is involved, among other things. If you could summarize what a macro "is" with one word, the word would be "automation."

To "automate" tasks you would do in WordPerfect, you can write one or more WordPerfect macros. Such automation is particularly useful to perform repetitive tasks which you frequently do in WordPerfect. Because the list of things you can in WordPerfect is endless and is only limited by the perception of a particular user, the list of things you can do with macros is endless, as well. But, as a beginning point, just know this: Anything you can do in WordPerfect manually can be automated by one or more WordPerfect macros. And, as noted with the Child Support Computation example just mentioned, macros also give you the possibility of doing some things you could not do in WordPerfect without the use of macros – or, at least, if you could, it would be very very hard to accomplish without the use of macros.

- **STORAGE LOCATION OF YOUR MACROS.** Since WordPerfect macro files are WordPerfect "files", they are located somewhere, usually on your hard disk. As a preliminary matter, it's good to give a little thought to disk organization – where do you plan to keep your macros? If you plan to keep them all in the same place, it's probably best to keep them all in your WordPerfect [default macro directory](#), shown below. But, you might want to create one or more particular directories for particular kinds of things. For example, in my *Grande Macros* program for Oklahoma lawyers, which consists not only of numerous interconnected macros but interconnected regular WordPerfect documents also, it made sense to create a particular dedicated directory, C:\Grande5, in which to store all interrelated macro and document files.

The main thing is to have and know the location of where you keep your macros, even if it is simply an acceptance of the organization that WordPerfect has made for you in the first place. To edit a macro, you need to know where it's stored and you need to know its name – else you wouldn't know what file to open. If you make interrelated macros (macros which link to each other when they run), in my opinion it is best to use the full pathname of the target macro in the macro that runs it, particularly in post-WordPerfect 10 versions. If you use the full pathname of the target macro, you will never be harmed, but, you may be if you don't. If all your macros are in WordPerfect's default macro directory or default supplemental macro directory, then you can use the [?PathMacros](#) or [?PathMacrosSupplemental](#)

system variable as the pathname part of the file you want to open. If you store particular macros elsewhere, you can create a global variable which identifies that path, e.g., Global vPath="C:\Grande5\". Then, to identify a macro to run from a macro that is already running, the statement might be Run(vPath+"thatmac.wcm"). Note that if you use a specific path other than ?PathMacros or ?PathMacrosSupplemental in identifying part of the pathname, the literal pathname is not fluid and the macros and/or related documents must actually be in the specified directory – otherwise, target macros will fail. J. Dan Broadhead offers the following suggestions:

I have a suggestion that we usually made to macro users concerning the "Default" and the "Supplemental" macro locations. I think WP used to be set up this way, but somehow got changed along the way.

As you mentioned, I believe that, initially, WP is now configured so that the "Supplemental" macro directory is blank, and the "Default" macro directory points to the installation directory of the Shipping Macros.

I would suggest a change here. Move the "Default" installation directory location to the "Supplemental" location field. Then create your own personal macro directory someplace, like in "My Documents\My WP Macros" (or something), and put that location in the "Default" field.

Here's why. It's a good idea to keep "your" macros separate from the WP macros. If they are stored together, then when you go to edit your macros, you might accidentally (or on purpose) edit and change one of the standard shipping/included macros that came with WP, and you might "break" it.

If your macros are kept separate, then when you open a file open or save dialog for your macros, then your own personal macro folder is displayed containing only your own macro files, and not the standard WP macros.

Every time you play a macro, WP will first look in the "Default" location, and then the "Supplemental" location till it finds the macro, so it can still find the standard macros.

All your macros will now normally be stored in your new personal macro folder, and if this is in the Windows "My Documents" folder someplace, then you can be sure that if you backup the "My Documents" folder on your computer, that you will get copies of all your macros as well. The idea behind the "My Documents" folder, is that all the things on your computer that you change, are all stored in this one place and you don't need to hunt all over your system for the files that you have changed.

If you want to change a shipping macro, then copy it from the standard installation directory (now pointed to by the "Supplemental" field) to your personal macro folder (the "Default" field), and make changes to the copy. Now when you run the shipping macro, it will first find and run your personally modified version instead, which is also in your personal macro folder someplace in "My Documents", and it will also get backed up when you save your "My Documents" folder.

If you mess up the shipping macro, it is only a copy, and you can delete the copy to get back to the original shipping version.

And for all the other file locations in WP that support both "Default" and "Supplemental" fields, I would do the same thing, pointing the "Supplemental" fields to the standard installation locations normally found in the "Default" fields, and pointing the "Default" fields to someplace in the "My Documents" folder. And do the same for PR and QP as well.

- **USUAL MACRO FILE LOCATION.** Unless you have changed the WordPerfect default macro directory, the default macro directory was established when you installed any Corel WordPerfect version. Those directories are:

WordPerfect 6.1	C:\Office\WpWin\Macros
WordPerfect 7.0	C:\Corel\Office7\Macros\WpWin
WordPerfect 8.0	C:\Corel\Suite8\Macros\WpWin

WordPerfect 9.0	WordPerfect 9 varies. If you had an early release, through Service Pack 2, it will be in your C:\Program Files\ directory shown below, but if you have the later versions, Service Pack 3 or 4, it will be in the C:\Windows\ directory shown below: Early versions (lower than release 9.0.0.738): C:\Program Files\Corel\WordPerfect Office 2000\Macros\WpWin Later versions (release 9.0.0.738 and higher): C:\Windows\Application Data\Corel\PerfectScript\9\WordPerfect
WordPerfect 10.0	C:\Documents and Settings\Owner\Application Data\Corel\PerfectScript\10\WordPerfect C:\Windows\Application Data\Corel\PerfectScript\10\WordPerfect (supplemental)
WordPerfect 11.0	C:\Documents and Settings\Owner\Application Data\Corel\PerfectScript\11\WordPerfect C:\Program Files\WordPerfect Office 11\Macros\wpwin (supplemental)
WordPerfect 12.0	C:\Documents and Settings\Owner\Application Data\Corel\PerfectScript\12\WordPerfect C:\Program Files\WordPerfect Office 12\Macros\wpwin (supplemental)

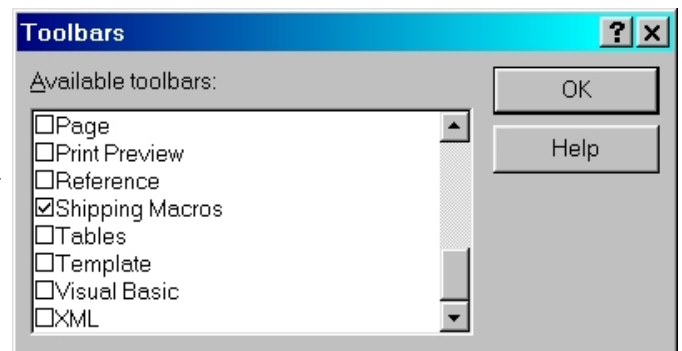
To know what your "release" version of WordPerfect is, click Help|About WordPerfect... and the specific release number will be stated.

To know (or change) your default macro directories in Wp8.0 and higher, select Tools | Settings... . In WordPerfect 6.1 or 7.0, select Edit | Preferences... . The Settings Dialog opens. Click on the Files icon which opens the File Settings dialog. Click the Merge/Macro tab. You will see two default macro directories, the Default and the Supplemental Macro directories.

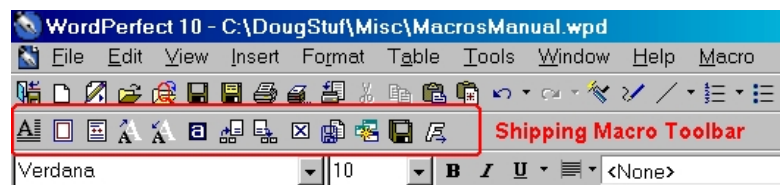
Subject to what was said in "J. Dan Broadhead's Tip", above, it is OK to store all your macros in the default macro directory/directories (although that will "clutter up" the original directory's contents with your own macros) or in some other directory you specify (which would probably be better). For example, the macros in my Grande Macros 6.0 program are all stored in C:\GrandeMacros.

- **SHIPPING MACROS.** Some macros, called "Shipping Macros", are installed during your installation of Corel WordPerfect (any version), unless you chose not to install them during the installation process. They would be installed in the "default" macro directory for the version of WordPerfect you are running.

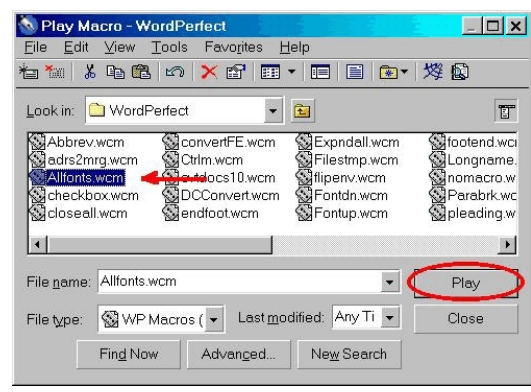
WordPerfect 9.0 and higher even have a special "Icon Toolbar" that you can use to play the Shipping Macros. To turn on this special toolbar, click View|Toolbars... and select Shipping Macros from the list, as shown here. The WordPerfect 10 toolbar looks like this:



Explanation/discussion about Shipping Macros is beyond the scope of this manual, even though I'll discuss a couple on the next page. To see what they can do, open the Shipping Macros Toolbar in WordPerfect 9.0 or higher, as shown above. Hover your mouse over an icon in the toolbar for a brief description of what the macro represented by the icon does, and, if you're interested, click it!



A shipping macro common to all WordPerfect versions is the macro file, "Allfonts.wcm." It creates a WordPerfect document which shows you ALL fonts installed on your computer, for your information. To run this macro, press Alt+F10 or click Tools, Macro..., Play, and from the Play Macro dialog select the Allfonts.wcm macro, shown below in much reduced size, just readable enough to make the point:



A "snippet" of the document produced is:

```

You have 276 fonts available to the Brother HL-1240 series:
Abadi MT Condensed Light Regular
aAbBcCdDeEffGhIiJkKlMmNnOpPqQrRsStTuUvVwWwXxYyZz1234567890~!@#$%^&*()_+-=;:~?/|<>[]{}

Allegro BT Regular
aAbBcCdDeEffGhIiJkKlMmNnOpPqQrRsStTuUvVwWwXxYyZz1234567890~!@#$%^&*()_+-=;:~?/|<>[]{}

AmerType Md BT Medium
aAbBcCdDeEffGhIiJkKlMmNnOpPqQrRsStTuUvVwWwXxYyZz1234567890~!@#$%^&*()_+-=;:~?/|<>[]{}

Arial Bold
aAbBcCdDeEffGhIiJkKlMmNnOpPqQrRsStTuUvVwWwXxYyZz1234567890~!@#$%^&*()_+-=;:~?/|<>[]{}

Arial Bold Italic
aAbBcCdDeEffGhIiJkKlMmNnOpPqQrRsStTuUvVwWwXxYyZz1234567890~!@#$%^&*()_+-=;:~?/|<>[]{}

Arial Bold Italic
aAbBcCdDeEffGhIiJkKlMmNnOpPqQrRsStTuUvVwWwXxYyZz1234567890~!@#$%^&*()_+-=;:~?/|<>[]{}

Arial Narrow Bold Italic
aAbBcCdDeEffGhIiJkKlMmNnOpPqQrRsStTuUvVwWwXxYyZz1234567890~!@#$%^&*()_+-=;:~?/|<>[]{}

```

The "real" and "full" document produced by this macro includes each installed font, together with a three-line presentation of each font. Why run this macro? To see what you have – chances are good you have installed fonts that you weren't even aware of. You can print the document for later reference.

Other Shipping Macros usually include: endfoot.wcm (converts endnotes to footnotes), footend.wcm (converts footnotes to endnotes), filestmp.wcm (inserts the filename in a footer on that page); reverse.wcm (which created this white-on-black text box; and several others.

- **HOW MACROS WORK.** When a macro "runs", a WordPerfect macro generally works the way WordPerfect does, top to bottom, responding to various codes as they are encountered (e.g., formatting) and in a lineal fashion. However, many twists and turns can occur depending on how the macro document is constructed. But, unless a macro is "told" otherwise (by including specific macro commands which do the "telling"), a macro starts at its top and progresses "downward" through the macro code until the macro is done. As each macro command is encountered, it is executed.

But, you will want to include commands which vary this "lineal" progression to fit your needs. You are the traffic cop – you tell the macro what to do, where to go, etc., by including various "traffic signals", so to speak, in the macro itself. See Chapter 4.

Think of a macro (for a moment) as though it were a "city." A city has predefined locations (street addresses). In macro vernacular, each such "street address" (location) is a "Label." A label is a "place" in the macro, that place being defined by you when you use the "label" statement. In conjunction with "Go" and/or "Call" commands, you can tell a macro to "go" to a particular label and execute commands at that location, shifting the "lineal" flow to a different location. Or you can "call" the commands contained at a label, instructing your macro to execute the commands within that called label *before proceeding* in the lineal flow. This is developed more fully in Chapter 4, Controlling Macro Flow, below, but, for now, understand that you can tell a macro to interrupt its "lineal" movement either temporarily or permanently. As I said, "you are the traffic cop" and you have the unfettered ability to design the macro any which way you want! So, while "structure" is indeed of great importance in writing macros, it might be helpful to know that *you are the authority* which determines what that structure is – WordPerfect's macro language does not place constraints upon you in that process, you are the person who imposes whatever constraints you might choose upon your particular macro creation. As the "creator", you call all the shots, even though you must abide the macro language's "rules" in doing so.

The bottom line: YOU determine "how a macro works." There is no "preordained" way. While you must conform to the very precise macro language's writing requirements, you can design a macro to do whatever your needs, interests and imagination, can conceive.

- **PLAYING MACROS.** Different methods exist. The main methods are described here.

! Using the Menu. Press Alt+F10 or click Tools| Macros | Play.... Either method opens the "Play Macro" dialog. Unless you've been using a different macro directory during a WordPerfect session, the **default Macro Directory's** contents will be shown. Find the macro you want, select it, and click the "Play" button.

! Using the Icon Toolbar. If you include a macro on a WordPerfect Icon Toolbar, all you need to is click the icon to run that macro. You can edit any WordPerfect Icon Toolbar (the bar at the top of WordPerfect with all the little pictures in it) or you can make your own, e.g., a toolbar with nothing BUT macros on it. This manual doesn't show you how to edit a Icon Toolbar, but it's not hard to do. For exact visual instructions if you are internet ready, go here: www.dougloudenback.com/wp/toolbar/index.htm

! Using Your Remapped Keyboard. You can "map" your WordPerfect keyboard so that (for example) Alt+S or Ctrl+Alt+S or Alt+Shift+Ctrl+S) where "S" is the letter you want to use) plays a macro. Since some "letters" in combination with the Alt, Ctrl, Shift keys are already used for other purposes, they will be unavailable unless you map your keyboard – and here's a nice tip from Barry MacDonnell: <http://home.earthlink.net/~wptoolbox/Tips/Assign.html>.

For macros, features, programs, and keystrokes: You can use "reserved" keys such as Alt+T (which normally accesses the Tools menu). But if you do, the macro (or feature or program or keystrokes) will play instead of the Tools menu opening. However, you can still get at the reserved keys: If you press Alt, release it, and then press T (i.e., press the keys in sequence, not simultaneously), the Tools menu will open.

This manual doesn't show you how to edit a Keyboard to include a macro, but it's not hard. For exact visual instructions if you are internet ready, go here: www.dougloudenback.com/wp/keyboard/index.htm.

! Using Shortcut Keystrokes. When naming a macro, you can use simultaneous keystroke combinations, such as Ctrl+1, Ctrl+Sft+1 as the macro's name (e.g., to play "Ctrl1.wcm" or "CtrlSft1.wcm"). To play such a macro, you press the same keystrokes simultaneously. See **Using Shortcut Names**, described in Chapter 3, below. Unlike in the good ol' Wp5.1 DOS days, when naming the macro you cannot use your "Alt" key to name the macro (e.g., Alt+G), even though you can remap your keyboard to use available Alt+keystroke as mentioned above.

! Using Startup or Desktop Macro Links. If you want, you can make desktop (or other) shortcuts which link to a specific macro. Theoretically, if such a link is clicked, WordPerfect should then open and immediately run the macro. However, in practice, that usage does not always work satisfactorily – particularly if the macro is large and complex. In such instances, it can cause WordPerfect to lock up. My recommendation is to NOT do that UNLESS the macro is short and sweet. That said, using Desktop Links to "[filename].wpd" doesn't have the lockup problem. So, should you find the need, you could make a WordPerfect document WHICH contains links to macros in the document. The Desktop Link could be to THAT document, and the macros linked within THAT document could then be safely run. This is just an idea for you to consider. I do it with my Oklahoma Lawyer macro program and it works well.

NOTES

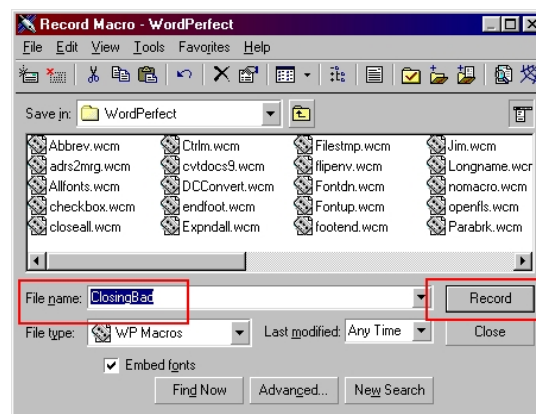
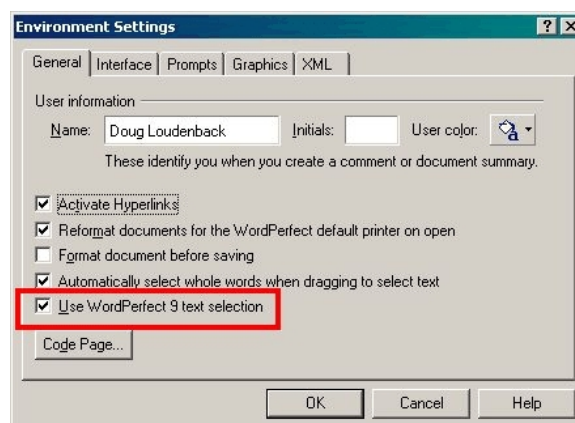
Chapter 3

First Things About Writing WordPerfect Macros

Links: The **Chapter Name**, above, moves to the next chapter. 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.

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				
Wp10's Select Method	Wp11 Problem	Writing Using Keyboard	Keyboard Shortcuts	A Dumb Example
				Writing From Scratch

- WORDPERFECT 10 And higher "SELECT" METHOD.** With WordPerfect 10, the default text selection method changed. Barry MacDonnell gives an excellent discussion of how this change can affect macro operation when using WordPerfect 10 or later at <http://home.earthlink.net/~wptoolbox/WP9TSS.html> wherein he explains and warns how the change can affect your macros when using various Select... and PosWord... commands. If you're like me, you'll want to "turn off" WordPerfect 10 and higher versions' default which mimics Microsoft Word's text selection method. "Snippet" macro code is at his website which tells you a macro work-around. On your own computer, you might want to engage WordPerfect 9 text selection in your "Environment" Settings: Press Shift+F1, click "Environment", and in the Environment Settings dialog be sure the "Use WordPerfect 9 text selection" check box is "checked." Also, see Chapter 9's [wp9select.wcm](#).
 - WORDPERFECT 11.0.0.225 AND 11.0.0.300.** The initial release of WordPerfect 11.0 (11.0.0.225) screwed up something – I'm not sufficiently clear about what it was, but what you need to know is that some macro commands, e.g., "?date..." commands, which are initially compiled in 11.0.0.225 won't run correctly in 11.0.0.300 (Service Pack 1) and vice versa. Forced recompilation is necessary. You can open a macro add and delete a "space" and then recompile. See <http://www.dougloudenback.com/wp/wp11warning.htm>. You can download a macro which does this rather automatically (you select the directory and all macros within the directory, if any, will be forcibly recompiled) at <http://www.dougloudenback.com/wp/Get&Compile.exe>.
 - WRITING MACROS USING THE KEYBOARD.** Although nothing particularly complex can be made this way, the simplest of all ways to write a WordPerfect macro is by using the keyboard. This "player piano" approach will record every keystroke (or mouse click) you make until you stop recording. You actually "write" no macro code at all – WordPerfect does that in the background. And, as will be seen shortly, it's even the simplest way to *begin* writing a complex macro. Here are the main things you need to know:
- ! Ctrl+F10 Starts and Stops the Macro's Recording.** Or, if you want to do it the "slow" way, using the Menu, select Tools|Macro|Record... to start, and Tools|Macro|Record... to stop (since a macro is recording, ✓Record... will have a



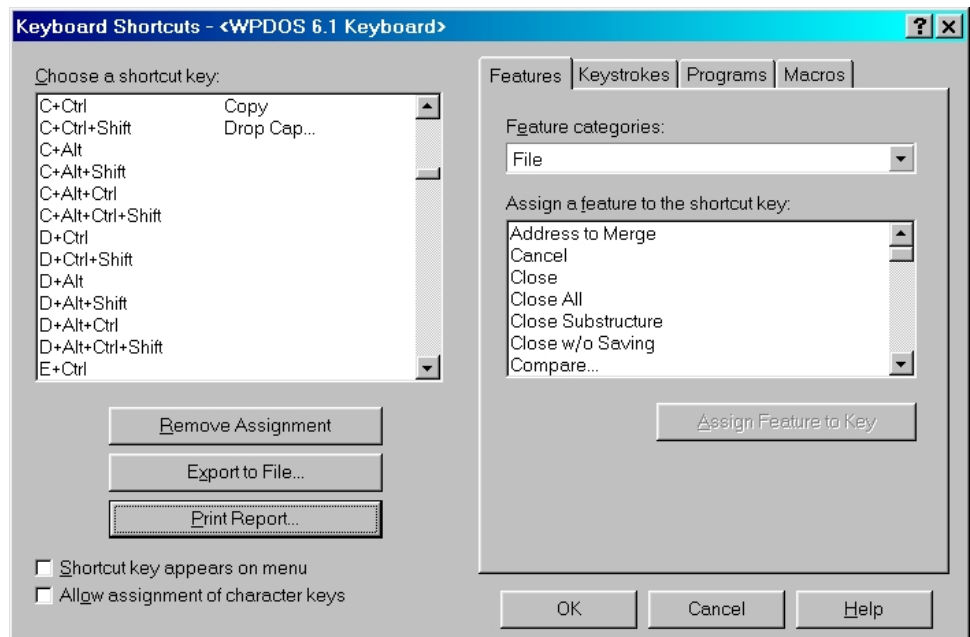
"check" by it, as just shown – clicking "✓Record..." stops the recording). As I said, simultaneously pressing Ctrl+F10 is lots simpler!

! Naming the Macro. When you press Ctrl+F10 to start the recording, the Record Macro dialog opens. Unless you have been using a different directory, it should open to your **default macro directory**. Assuming that's where you want to save the macro, enter a name in the File name box. It's best to use a name that makes sense in the context you'll be using the macro so that you'll remember it later. I've used "ClosingBad" for the name but you can name it what you want. WordPerfect will add the correct file extension, which is ".wcm". After naming the macro file, click the "Record" button. Then, the dialog closes and, from this point, any keystroke/mouse click you enter will be recorded until you press Ctrl+F10 again.

! USING SHORTCUT NAMES. You may want some often used macros to be played by pressing a simultaneous keystroke combination, such as Ctrl+1, Ctrl+Shift+1. WordPerfect 5.1 DOS users will recall using the Alt+key and other combinations with great abandon. You can use the Alt, Ctrl and Shift keys in combination with other keystrokes (but not Shift without Alt or Ctrl). All "numeric" keys are available, e.g., Ctrl+1 (for a macro named "Ctrl1.wcm"), Ctrl+Sft+1 (for a macro named "CtrlSft1.wcm"), etc. Many, but not all, "alphabetical" keys are available. To know what "alpha" keys are/are not available, take a look at your existing keyboard "map." In WordPerfect 6.1 and 7.0, click Edit|Preferences, and in WordPerfect 8 and higher, click Tools|Settings and select the Keyboard Tab. Select the keyboard you are using and click the Edit button to have a look. Note: Unless you have copied a keyboard in WordPerfect 6.1 and 7.0, you will not be able to click the Edit button for a "shipping" keyboard. But, you can copy that keyboard and then have a look at the duplicate you have made. I personally persist in using the old DOS keyboard, so here's a look at WordPerfect 10's Keyboard Shortcuts dialog:

Note that Ctrl+C (shown as C+Ctrl) is already used for Copy and Ctrl+Shift+C is used for Drop Cap. But, Ctrl+D, Ctrl+Sft+D and Ctrl+E are both available. Scroll through the list to see what naming possibilities exist. And, in WordPerfect 10 and higher, note that you can click the Print Report button for a 2 ½ page printout.

The exact syntax you must use when naming a shortcut keystroke combination to play a macro is this: Ctrl is always 1st; Sft (using those 3 characters) are always 2nd, if you're using the Shift key at all; the numeral/letter are always last. Ctrl+Sft+3 is correct, but Sft+Ctrl+3 and Ctrl+Shift+3 are not. The corresponding macro name would be "CtrlSft3.wcm" (case is unimportant).



! A DUMB KEYBOARD MACRO EXAMPLE. Using what was said in "Name the Macro", above, for the macro ClosingBad.wcm I've typed the following text:

Working with you in this matter has been one of the most exasperating experiences of my lifetime. While I certainly wish you the very best in all your future endeavors, it is my fervent wish than no such endeavor will involve me in any way.

Best regards,
Douglas Loudonback

Now, having done the typing, press Ctrl+F10 to stop the recording. The simple macro is done.

As will be shown later, any macro you make can be opened as a file so that the macro code can be viewed and/or edited. Doing that, the macro code for ClosingBad.wcm will look something like this:

```
Application (WordPerfect; "WordPerfect"; Default!; "EN")
Type (Text: "Working with you in this matter has been one of ")
Type (Text: "the most exasperating experiences of my lifetime. ")
Type (Text: "While I certainly wish you the very best in all your ")
Type (Text: "future endeavors, it is my fervent wish that no such ")
Type (Text: "endeavor will involve me in any way.")
HardReturn ()
HardReturn ()
Type (Text: "Best regards,")
HardReturn ()
HardReturn ()
HardReturn ()
Type (Text: "Douglas Loudenback")
```

The 1st line of code is the "Application" statement which identifies the application in which the macro was written and/or will be used. The closing "EN" identifies the language version, in this case, "English." In WordPerfect 6.1 and 7.0, four "English" versions existed, one of which was "US", and the Application statement ends with "US" for the USA versions of Wp6.1 and 7.0. Actually, if you don't intend to write macros which work in other Corel WordPerfect Suite programs, the line has been unnecessary since WordPerfect 8. Certainly, it does no harm and it is automatically inserted. **If you are using WordPerfect 6.1 or 7.0, the line will read a bit differently.**

After the 1st line, everything done during the macro record phase has been translated into macro code – some of which is unnecessary for the macro to function. The () following HardReturn codes has no importance, nor does the "Text:" in the Type (Text:) commands. Type ("Working with you in this matter...") would do just as well.

Now, to insert the above text at the end of a "real" letter, **run the macro** ClosingBad.wcm and the text will be typed at the insertion point.

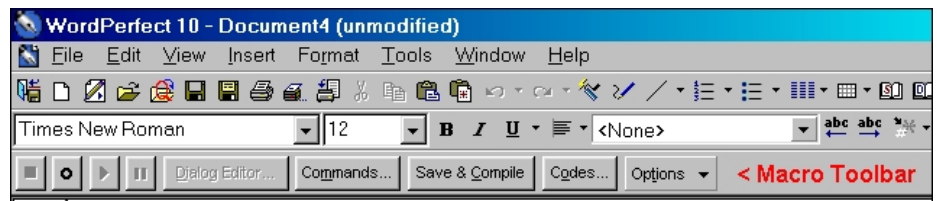
This example is probably too dumb to be useful other than to make the point: You can use the Keyboard to write any text document or portion of a text document that you may want – simple or complex. The document could be a repetitive form, standard paragraph, whatever.

- **WRITING MACROS FROM SCRATCH.** If you don't want to start writing a macro from your keyboard, described above, but want to "just do it" from scratch without first naming the macro and merely enter macro commands from scratch, you need to know and do a few things. Note that these things are done automatically by WordPerfect when you open an existing macro for editing. But, if you're doing it from scratch in a blank WordPerfect document, there are two ways to go – (1) make a macro "shell" and then edit the macro as needed; or (2) start totally from scratch.

! Make a Macro Shell and Then Edit the Macro. For me, this is the simplest way to go. (a) press Ctrl+F10 to start recording a new macro; (b) name the macro in the Record Macro dialog which opens; and (c) type anything ... a space, a hard return, whatever; (d) last, press Ctrl+F10 to stop recording. Then, a shell of your macro is made. You can open the macro to begin your "real" editing – of course, when you do, delete the keystroke(s) you entered from the keyboard in step (c).

! OR, Do It Totally From Scratch. First, turn on the Macro Toolbar– Tools|Macros|Macro Toolbar. This is a special toolbar for use in compiling WordPerfect macros, and, if needed, inserting particular

codes into the macro document. Although the appearance will vary slightly with WordPerfect versions, it will always appear immediately above the document's white space it will always pretty much look like the toolbar at the bottom of the image shown above.



Turning on the Macro Toolbar does these things: Line numbering is turned on – this will prove helpful when receiving compilation error or warning messages when the macro is compiled; "Quick" stuff is turned off – QuickCorrect, QuickWords, Speed Links, Format-As-You-Go, SmartQuotes. That's what you want – "curly" quotation marks won't work in macro commands that need quotation marks. When you're done with editing your macro, these features should automatically revert to the way they were before you turned on the Macro Toolbar.

! Be Sure That All "Quick..."Codes Are Turned off – QuickCorrect, QuickWords, SpeedLinks, Format-As-You-Go, SmartQuotes. While this should all be done automatically when you turn on the Macro Toolbar, if it doesn't, use Tools|QuickCorrect... and make sure that all the boxes in each of the named Tabs are "unchecked." Macro code does not recognize curly quote marks and some other stuff as valid characters. Compilation errors result if you don't turn all this stuff off. As noted above, all this is done by WordPerfect automatically if you open an existing macro for editing. But, here, the context is that you are "starting from scratch." Turning on the Macro Toolbar should automatically disengage these items. Be sure that it does – turn on Reveal Codes, type a regular quotation mark ("), position the insertion point immediately before it, and look at it in Reveal Codes. If the character is identified as being from a character set, probably 4,32, then it's a "curly" quote, meaning that SmartQuotes is still turned on, and, if so, turn them off manually.

! Then, Write Your Macro Code. If you are writing a WordPerfect 6.1 or 7.0 macro from "scratch", it may be desirable (but not necessarily required) to have this line at the top of the macro: **Application** (A1; "WordPerfect"; Default; "US"). For WordPerfect 8 and higher (if you use the Application statement at all, unneeded in those version), the top line should be: Application (WordPerfect; "WordPerfect"; Default!; "EN"). Notice the variances between Wp6.1/7.0 and Wp8.0 and higher – the "A1" parameter is not used in the later versions and the "!" following "Default" is not used in Wp6.1 or 7.0 but is used in Wp8.0 and higher. See **Writing Macros During Macro Editing**, below, for more information about writing the macro. It's the same whether doing a macro from scratch or while editing an existing macro.

! Save & Compile the Macro. When done, click the "Save & Compile" button in the middle of the Macro Toolbar. After compilation is done, close the file (the macro is compiled when the "New Blank Document" icon on the Icon Toolbar is white, not dim, and the Save & Compile button is dim, not black).

The particular code you write and use of the Macro Toolbar is more particularly described in other portions of this document.

NOTES

Chapter 4

Controlling Macro Flow

Links: The [Chapter Name](#), above, moves to the next chapter. 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.

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				
Quit	Go	Call	Return	
Label	OnError	OnNotFound	OnCancel	

This short chapter describes the main commands/statements a common macro user needs to know to control the "flow" a macro takes during its operation. As noted in [Chapter 2](#), macros generally run "lineally", top to bottom, and they do so unless you enter commands which instruct differently.

- ! QUIT** The [Quit](#) command totally stops any/all macros from running whenever it is encountered. Usage: Type the word Quit in the macro.
- ! GO ()** The [Go](#) command tells your macro to "go to" a different [label](#) (location) within the same macro. The parameter is the name of the referenced label. Usage: Go(NextLabel).
- ! CALL ()** [Call](#) "sucks up" (sorry ... executes) the commands contained in and after the label name – the parameter – identified in the Call command. At some point after the called label, a Return command "returns" execution to the point in the macro immediately following the Call(NextLabel) command. Usage: Call(NextLabel) – however, a "Call" command is implied when merely used with the label name, so, NextLabel is the same as Call (NextLabel).
- ! RETURN** Within a macro, the [Return](#) command concludes a Called statement/ callback/routine. When encountered, macro flow "returns to" and resumes at the exact point in a macro following the Called statement/callback/routine. The Return command need not be included within the called label (i.e., one or more labels may follow the called label if there is a need/reason to do so), but, unless Return is encountered somewhere, macro flow will never return to the point in the macro which contained the Call command. In the context of the Run (Macroname) command, in which one macro "runs" another, Return concludes the macro being run and returns macro flow to the macro immediately following the Run (Macroname) command. Return can be accompanied with optional parameter, the enumerations of which are CancelCondition!, ErrorCondition!, and NotFoundCondition!, not covered in this manual.
- ! LABEL ()** [Label](#) isn't a command but is a "place", an address, in a macro. The "name" (the parameter) of the address is contained within parentheses, as follows: Label(NextLabel). It is ordinarily used as the destination address in statements such as Go(NextLabel) or Call(NextLabel). A label name must begin with a letter but can contain letters, numbers, or "_" after the initial letter. Case is not important when naming a label or when referencing it, so, Go(NextLabel) is the same as Go(nextLabel).
- ! ONERROR ()** If a macro operation "error" is encountered when a macro is running, it ordinarily stops the macro and some sort of error message dialog will be present on-screen. [OnError](#) can be used to prevent that, particularly when you can anticipate that a user might make an error in data entry when a macro is running. Use an [OnError](#) command to direct macro flow to the name of the label identified in the OnError statement, e.g.,

OnError (NextLabel). 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. An OnError statement without a label-name parameter turns off all OnError statements that may be effective before that time. Usage: OnError (NextLabel). An alternative: **OnError Call ()**.

! ONNOTFOUND () If a macro contains search routines, and if a search routine fails to produce a match, then an on-not-found condition occurs, the macro will stop, and a message dialog will appear on-screen. To prevent that, an **OnNotFound** statement tells the macro to go to the label identified in the OnNotFound parameter, e.g., OnNotFound (NextLabel). An alternative: **OnNotFound Call()**.

! ONCANCEL () A macro will stop running when a cancel condition occurs unless it is preceded by an **OnCancel** statement. A cancel condition is not the same thing as a **Quit** command but includes such events as a Quit button or a Close button being clicked in a dialog. OnCancel(LabelName) tells the macro where to go if a cancel condition occurs and macro statements in that **label** will be executed. An alternative: **OnCancel Call()**, e.g., OnCancel Call (NextLabel) - but remember to use a Return command at the end of the called label.

Here's a simple example that uses most of the above commands/statements – no colored items are links but are colored/bolded for emphasis – **Labels** and **Commands** – if you want to copy this macro, give it a name, e.g., Test.wcm - color/bold/other formatting codes don't matter either way, yes or no:

Application (WordPerfect; "WordPerfect"; Default!; "EN")

Label(vBegin)

OnCancel (vSure)

MessageBox(x;"Hi - this is the starting point of this macro";"This macro don't REALLY do nuttin!";OkCancel!)

If(x=2) Assert (CancelCondition!) EndIf

// in the above, if x=2, e.g., the Cancel or Close button, a Cancel Condition is emulated

Label(vNum)

GetString(x;"Type a real number, not a letter";"Please enter a number to test")

OnError (vNotNum) x=StrNum(x)

MessageBox(;"Great! You typed a number!";"You typed "+x)

MessageBox(y;"Do you want to search for "+x+" in the current document?";

"If you want to see if "+x+" is present in the current document, click 'Yes'";YesNo!)

If(y=6) Call(vSearch) EndIf

Call (vAgain) If(x=6) Go (vBegin) Else Quit EndIf

Label(vAgain)

MessageBox(y;"Do you want to do the macro again?";

"If you want to start at the beginning, click Yes."; YesNo!)

If(y=6) Go(vBegin) Else Go(vEnd) EndIf Return

Label(vSearch)

OnNotFound (vSorry) If(?DocBlank) Assert(NotFoundCondition!) EndIf

PosDocTop SearchString(x) MatchPositionAfter SearchNext

MessageBox(;"Paydirt!";"Yes, "+x+" is in the open document") Return

Label(vSorry)

MessageBox("Sorry!";x+" is not in the open document") Return

Label(vNotNum)

MessageBox(x;"Sorry, you did not enter a number";x+" is not a number. Do you want to try entering a number again?";YesNo!) If(x=6) Go(vNum) Else Go (vEnd) EndIf

Label(vSure)

OnCancel

MessageBox (x;"Are You Sure You Want To Cancel This Macro?";

"You clicked the Cancel button in a prior message box. Are you sure you want to quit?";YesNo!)

If(x=6) Quit EndIf Go (vBegin)

Label(vEnd)

MessageBox("We're done with this macro!";"It's been swell!") Quit

[Top of Chapter 4](#) [Chapter 5](#)

NOTES


Chapter 5

Using the Dialog Editor During Macro Editing

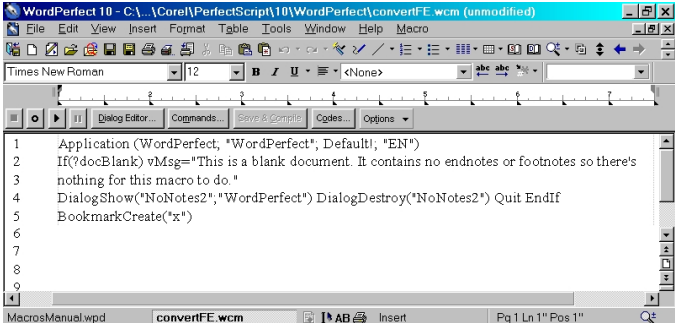
Links: The [Chapter Name](#), above, moves to the next chapter. All page header links go to the contents menu. Within the chapter, [Topic Titles](#) returns here. [Other Red Links](#) are to other locations in this paper. [Blue Links](#) are to web sources.

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					
Opening a file	Editing Environment	Macro Toolbar At a Glance	Macro Toolbar "Little Buttons"	Using Stop and Record Buttons	The Commands Button
Macro Toolbar Codes Button	Macro Toolbar Dialog Editor Button	Copying A Dialog To Text	Copying A Dialog To Dialog Editor	Innards of Dialog Editor Dialog	Dialog Properties
Dialog Font	Changing Dialog Properties	Dialog Controls	Adding Controls	Control Names	Control Size and Placement
Associated Variable	List Boxes	Static Text Boxes	Push Buttons	Combo Boxes	Viewer Boxes
Bitmap Boxes	Date Boxes	Radio Buttons	Check Boxes	Edit Boxes	
Alignment of Controls	Ordering the Controls	Controlling the Sequence	Default Button	Initial Focus	Close and Save the Dialog
Enter Macro Code		Macro Toolbar Save & Compile Button		Compilation Errors	

As noted in [Writing Macros Using the Keyboard](#), above, making the beginnings of a macro by using Ctrl+F10, naming the macro, and then doing something simple ... such as typing a space ... and then pressing Ctrl+F10 to stop the macro, is a good way to begin to write a macro that you eventually intend to edit and make more complex. Doing so (a) automatically **turns off the Quick... stuff** which needs to be off when macro code is made, and (b) sets up the **"Application"** statement correctly and automatically. In short, that process creates a quick "shell" of a macro which you can then edit to make as varied as you want. This section intends to acquaint you with what you see and some of the things you can do when you edit a WordPerfect macro.

! FIRST, OPEN A MACRO FILE FOR EDITING. Since a WordPerfect macro file is, in its visible form, merely a souped up regular WordPerfect document, open the file you want to edit just as you would any other WordPerfect file. You can use regular File|Open... routines or use Tools|Macro|Edit... to open a macro file, it doesn't matter. See [Storage Location of Your Macros](#) and [Usual Macro File Location](#) as to where to find macros to edit. A macro file is identified by its file extension, ".wcm", and by its icon – a diagonal audio tape cassette: . If you haven't changed your default Windows environment (which does not show a "registered" program's file extensions), you won't see the ".wcm" but you will see the icon.


! EDITING ENVIRONMENT. When a macro file is opened, QuickCorrect, QuickWords, SpeedLinks, Format-As-You-Go, SmartQuotes settings are automatically turned off; the document opens in "draft" view; and the Macro Toolbar appears at the top of the document. Scroll into the left margin area and notice that all lines are consecutively numbered, a useful feature when compiling a macro which, on compilation error, reports the approximate line number containing the error, plus other information. The view will be something like this picture.



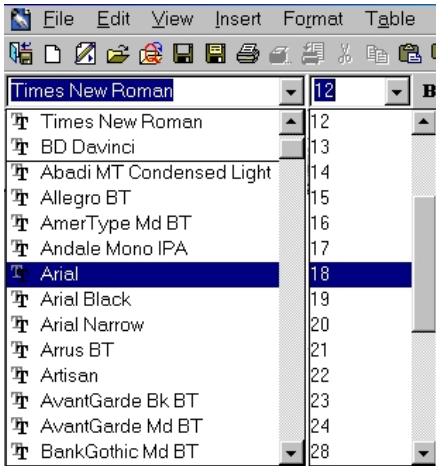
! MACRO TOOLBAR AT A GLANCE. The Options button receives no discussion in this manual. See on-line Help for more about it.

The other buttons are discussed below.

! THE "LITTLE" BUTTONS. P Stop, N Record, | Play, and ~ Pause.

	These 4 small buttons are: Stop Record Play Pause and are explained below.	This button accesses the Dialog Editor, used to make or edit Dialogs.	This button accesses the Command Browser. Use it for quick macro help information or to insert commands.	When done editing, click the Save & Compile button to compile the macro.	Use Codes to insert Wp code that can't be typed in for Search & Replace code, as you do in Wp docs.	I never use the Options button. It won't be discussed in this manual.
---	---	--	---	---	--	--

Two of these (P Stop and N Record) are quite handy for inserting into your macro document the commands representing something you do using the keyboard or mouse. While you certainly can't do everything with these features, you can do a lot. For example, let's say at a particular point you want to insert a different Font and Font Size. While you could enter the code manually, it's easier and quicker to enter the code using the N Record button. So, click the N button to start the recording. That immediately opens an empty document. On the Property Bar, I've selected Arial as the font and 18 as the size, using the mouse pointer to do so:



Having done that, click the P Stop button (it is covered up in this picture by the drop-down font list box) to stop the recording. The regular macro document will again be visible. The following code is now present into the macro document:

Font (Name: "Arial "; Family: 16662; Attributes: FontMatchNormal!; Weight: 90; Width: WidthUnknown!; Source: DRSFile!; Type: TrueType!; CharacterSet: FontMatchASCII!)

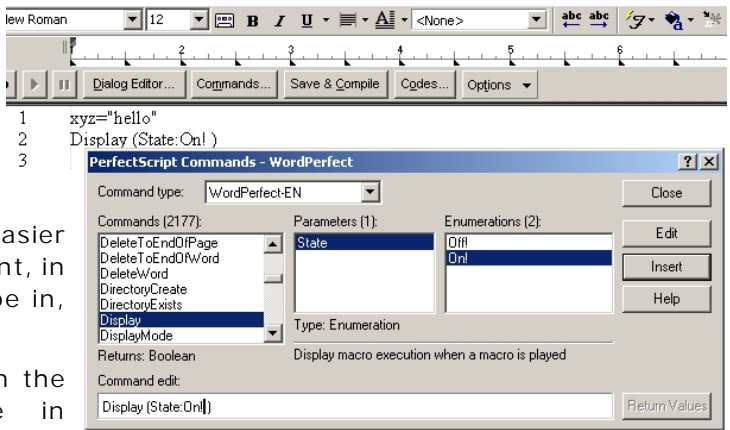
FontSize (FontSize: 0.25")

This method may be easier and quicker than entering code manually. When using this feature, the unnecessary-to-operate but helpful-to-understand descriptive parts of the code are inserted, e.g., Font (Name: ...) – the text "Name:" being unnecessary.

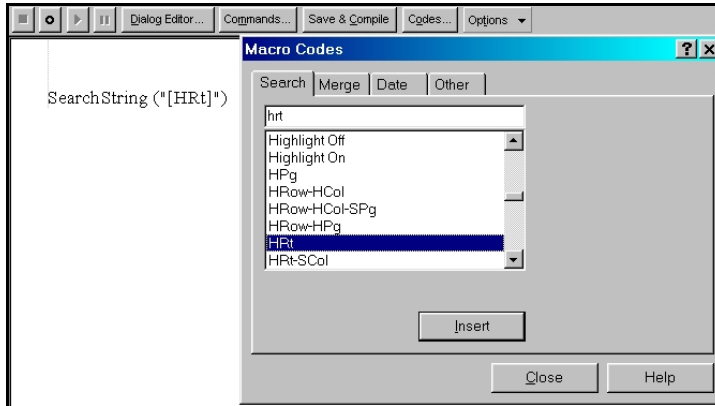
I have little use for the | Play and none for the ~ Pause buttons. The play button runs the macro in the macro document but if the macro writes to the currently open document, that document *is the macro* itself – you may shoot your macro in the foot! Also, macros often run sluggishly if "run" in the macro document itself. I don't use the Pause button or any Pause... commands at all. See on-line Macro help for more.

! THE COMMANDS... BUTTON aka the Commands Inserter button. It opens the Commands Browser/Inserter, discussed more completely in Chapter 1. While I consider the Command Browser's best use to be its quick access to information about macro commands, it can also be used to insert commands into the macro you are editing. Here's an example. It's easier just to type what you want in the macro document, in my opinion. But, if you don't know what to type in, this may be very useful.

Note that command syntax is 100% reliable in the Commands Browser, which is not true in WordPerfect's on-line macro help.



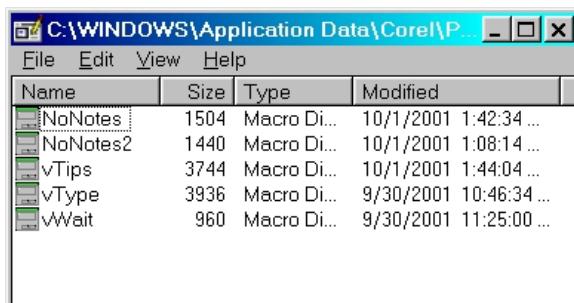
! THE CODES... BUTTON. If you are writing a search/replace operation in the macro which uses a WordPerfect "code" as part of the search or replace, merge, date or "Other" (Chpt#, Page#, SecPage#, TotPage#, Vol#), you can't "type in" that "code" using the keyboard. Use the Codes... button to insert the code. My personal use of this button is limited to Search/Replace commands. For example, if you want to search for a hard return, the [HrT] code would be inserted in the SearchString statement as is shown in this picture:



You supply the quote marks.

The other Tabs (Merge, Date, Other) are not explained in this manual. See on-line Macro help for information about them.

! THE DIALOG EDITOR BUTTON. Click the Dialog Editor... button to access the list of existing dialogs in the macro (that is, dialogs made with and in the Dialog Editor – dialogs written from the keyboard would not be in the Dialog Editor list). When you click the button, the list of dialogs opens – here, I'm showing the dialogs used in **ConvertFE.wcm**, the text of that macro being in **Chapter 9**, below.



Double-click on an existing dialog name to edit the dialog. Select File|New to make a new dialog. Or, with a dialog selected, select Edit|Copy to copy it and/or Edit|Paste to paste it into the list (it will be slightly renamed). To delete or rename a dialog, use File|Delete and File|Rename as needed. Dialog names are case sensitive, so, when using a dialog in the macro, the name of the dialog must exactly match, as with DialogShow("vTips"; "WordPerfect"). DialogShow("Vtips"; "WordPerfect") would not work.

! COPYING A DIALOG TO TEXT. If a dialog has been made using the Dialog Editor, the dialog can be converted to the macro's text (or simply in some other document) if you want. This cannot be done in WordPerfect 6.1 but can be done in WordPerfect 7.0 and higher. First, select a dialog to copy in the dialog list. Then, use Edit|Copy (or press Ctrl+C) which copies the dialog code into the Windows Clipboard. Then, somewhere in the macro (or other) document, Paste the code using the Paste Icon (or Ctrl+V). The literal macro code used in the macro will appear, together with a closing DialogShow command. For example, in the above list of dialogs, I'll select vType, press Ctrl+C, and then, in this document, press Ctrl+V, resulting in (except that I've changed the font):

```

1 DialogDefine (Dialog: "vType"; Left: 50; Top: 60; Width: 236; Height: 138; Style: Percent! +NoCloseBox!;
2 Caption: "Convert Endnotes or Footnotes To Text Notes")
3 DialogSetProperties (Dialog: "vType"; FontName: "Arial"; FontSize: 9p)
4 DialogAddListBox (Dialog: "vType"; Control: "B1"; Left: 64; Top: 48; Width: 43; Height: 20; Style: 0;
5 MacroVar: var0)
6 DialogAddListItem (Dialog: "vType"; Control: "B1"; Item: "Footnotes")
7 DialogAddListItem (Dialog: "vType"; Control: "B1"; Item: "Endnotes")
8 DialogAddListBox (Dialog: "vType"; Control: "B2"; Left: 64; Top: 73; Width: 43; Height: 20; Style: 0;
9 MacroVar: var1)
10 DialogAddListItem (Dialog: "vType"; Control: "B2"; Item: "Yes")
11 DialogAddListItem (Dialog: "vType"; Control: "B2"; Item: "No")
12 DialogAddListBox (Dialog: "vType"; Control: "B3"; Left: 195; Top: 66; Width: 29; Height: 63; Style: 0;
13 MacroVar: vSz)

```

```

14 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"1")
15 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"2")
16 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"3")
17 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"4")
18 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"5")
19 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"6")
20 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"7")
21 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"8")
22 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"9")
23 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"10")
24 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"11")
25 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"12")
26 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"13")
27 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"14")
28 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"15")
29 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"16")
30 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"17")
31 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"18")
32 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"19")
33 DialogAddListItem (Dialog:"vType"; Control:"B3"; Item:"20")
34 DialogAddPushButton (Dialog:"vType"; Control:"OKBtn"; Left:130; Top:118; Width:50; Height:12;
35 Style:OKBtn!; ButtonText:"OK")
36 DialogAddPushButton (Dialog:"vType"; Control:"TipsBtn"; Left:78; Top:118; Width:39; Height:12;
37 Style:0; ButtonText:"Tips")
38 DialogAddPushButton (Dialog:"vType"; Control:"CancelBtn"; Left:16; Top:118; Width:50; Height:12;
39 Style:CancelBtn!; ButtonText:"Cancel")
40 DialogAddText (Dialog:"vType"; Control:"Static1"; Left:4; Top:6; Width:228; Height:38; Style:Left!;
41 Text:"This macro will convert footnotes or endnotes to text, beginning at the top of the document. The
42 actual note number code will be replaced with a superscript number as text. Corresponding text notes will
43 be added in a ""Notes"" group at the end of the document.")
44 DialogAddText (Dialog:"vType"; Control:"Static2"; Left:3; Top:50; Width:56; Height:10; Style:Right!;
45 Text:"Convert what?")
46 DialogAddText (Dialog:"vType"; Control:"Static6"; Left:23; Top:76; Width:33; Height:10; Style:Right!;
47 Text:"Watch?")
48 DialogAddText (Dialog:"vType"; Control:"Static8"; Left:111; Top:48; Width:81; Height:10; Style:Right!;
49 Text:"Current font point size:")
50 DialogAddText (Dialog:"vType"; Control:"Static9"; Left:199; Top:47; Width:21; Height:12;
51 Style:ShadowBox!+Left!; Text:"")
52 DialogAddText (Dialog:"vType"; Control:"Static11"; Left:140; Top:70; Width:43; Height:37; Style:Right!;
53 Text:"Point Size to use for the Notes:")
54 DialogShow (Dialog:"vType")

```

Whether you'll ever want to do this, I can't say. But, if you are wanting to learn the structure associated with writing dialogs from the keyboard, this is a useful technique in learning how to do that.

Although I initially made the above "vType" dialog using the Dialog Editor, the above code would produce the **exact exemplary dialog** used in the explanations of the Dialog Editor which begin below – but note that dialogs made by keyboard text (like the above) cannot be seen/edited in the Dialog Editor. Only dialogs made IN the Dialog Editor can be seen/edited in it.

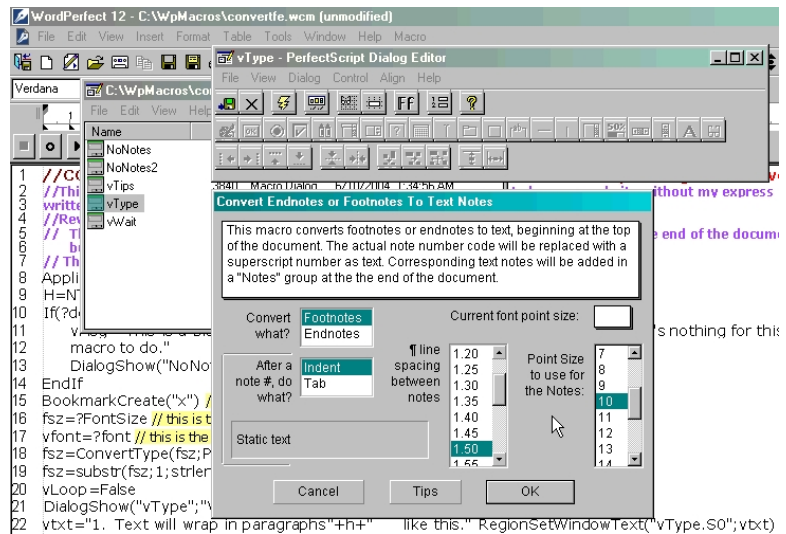
And, note that you may or may not want to use the closing code which is inserted at the close of such a Copy/Paste routine, the DialogShow("vType") command. Instead, you may want to insert such a command at some other place in the macro's text. For Example, the whole routine, beginning with the 1st Line, above, could be preceded by a Label() statement, such as Label(vDialog) and, instead of using the last Line, above, you could mark the end of the label with a Return command. Then, you could use Call(vDialog) elsewhere in the macro and use DialogShow("vType") following the point of the Call.

! Reversing The Process. Again, not that you'll ever want to do it, but it is also possible to effectively copy a dialog you've made with the **DialogDefine** keyboard method into the Dialog Editor using the DialogSave() command, not covered in this manual. See Julie Jeppson's June 1997 *PerfectScript Journal* issue at <http://www.wpmacros.com/psjournal/june97feat.htm>.

- IF YOU DO EITHER of the above, note that you may only have one dialog with the same name. Dialogs with the same name cannot coexist ... either delete the Dialog Editor dialog, or delete the dialog you made with **DialogDefine**. If you don't, error will result when you run the macro.
- **INNARDS OF A DIALOG EDITOR DIALOG**

Here, I've opened the dialog named "vType" to edit. It opens into the Dialog Editor which consists of 2 parts: (1) a floating Toolbar which you can drag to a location that suits you; and (2) the dialog you have chosen to open for editing. If you have a large dialog, the floating toolbar sometimes gets hidden behind it – but you can "maximize" the editor by clicking the "maximize" button just as you would with any other window.

This macro is available for download at <http://www.dougloudenback.com/wp/ConvertFE.exe> in a self-extracting Winzip file. Also, see **Chapter 9** for the code used in this macro.



Most buttons represent controls that are selected by a click, followed by another click somewhere in the dialog, thereby inserting a "model" of the control for you to edit. In the above, the "controls" include several "Static text" boxes - boxes that don't "do" anything other than contain descriptive notes about what a user should do, etc.; three "List boxes"; and three "Push Buttons." Any control you insert can be selected by clicking on it and then resized or moved by dragging it as needed. Once selected, a control can be copied or deleted, described below. Some of the "high points" you need to know are the following:

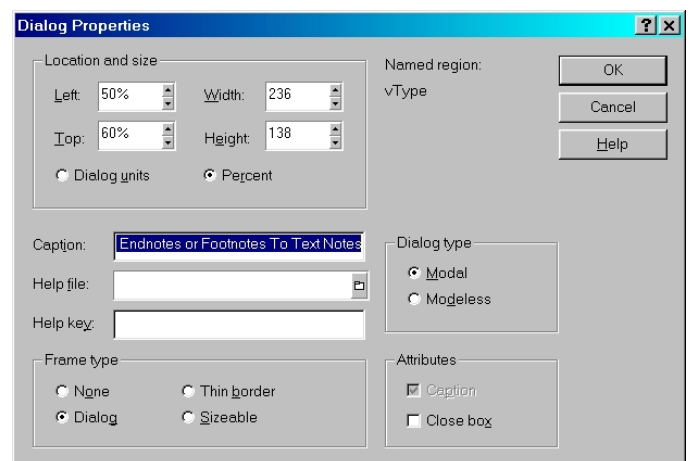
- ! **THE DIALOG'S PROPERTIES.** The dialog's name is whatever you name, or later rename, it to be. Remember that the dialog's name, when you use it in the macro, is case sensitive. Other properties are usually set in the Dialog Editor after you open it for editing. Having opened the dialog for editing and with nothing selected, right-click in the dialog to see the pop-up list shown in the left picture.



Right-click in the dialog to get this list.

Note the "non-properties" items: "Test" to test the dialog; "Control Order" to set the sequence of controls – when the Tab key is pressed, input focus moves to another control in the dialog; "Initial Focus" to set the input point when the dialog opens; "Default Button" to set the button which executes when the "Enter" key is pressed; among other things. But, back to the point.

Font: A font is automatically used when a new dialog is created, probably MS Sans Serif 8p. But, you can change that by selecting Font and set what you want.



Properties: To reset the dialog's other properties, select Properties... (or double-click on the dialog's title bar). Either method opens the dialog's Properties box. *Do you see the Help button in this dialog? Use it!*

Among the options, edit a title bar's text (caption) and set location and size. *To force the dialog's display to be in the center of the screen, set Left & Top values at 50%.*

Note the various possible Frame types. "Dialog" is usual, but try the others to see the options. By default, a dialog contains a "Close box" - the "x" at the upper right corner of the dialog. Usually, I don't want the

"x" so I "uncheck" the Close box, but [see this example](#) of its use in Chapter 8. "Dialog type" is "modal" or "modeless." If "modeless", a user can click elsewhere in WordPerfect while the dialog is displayed. Usually, that is unwise. Usually, use "Modal."

The Left, Top, Width and Height dimensions can be set here. Though less precisely done, that can also occur without using the Dialog Properties box by "dragging" the dialog: (1) click the dialog title bar; (2) hold down your mouse pointer on the title bar and "drag" it where you want. You can also "drag" on the dialog's borders and corners to resize it. If you do, and if you want the dialog's relative display to remain relatively constant, follow-up by opening the Dialog Properties box to be sure the relative position you want is present. Left 50% centers the dialog horizontally; Top 50% centers it vertically.

- **DIALOG EDITOR CONTROLS.** This explains some, not all, possible controls you can insert into a dialog. Various controls have different values to set. All controls have "names" – "control names"; and all have dimensional values of Left, Top, Width and Height. Most controls can be associated with a variable.

! Adding Controls. To add a control in a dialog, use the "controls" in the PerfectScript Dialog Editor toolbox – *the middle row of icons shown in this picture.* To add a



control to a dialog open in the Dialog Editor, single-click on the icon you want in the PerfectScript Dialog Editor toolbox; then single-click in the dialog itself. A model control be present at the point of your click for you to edit. Notice that the Toolbox contains other icons: 1st row – Save, Close, Test, Properties, Show/Hide Grid, Snap to Grid On/Off, Font, Order of Controls, and Help; 3rd Row – these are alignment icons, dim unless multiple items controls are selected – see [Alignment](#) later in the paper. The various "control" icons are shown below (I've rotated the bottom picture to identify control names).

All controls have these things in common:

Unique Control Name. Within a dialog, each control's name must be unique, and, if you are using callback routines that need to get the value in a control by using any RegionGet... command, the Control Names are case sensitive.

Size and Placement. Every control has 4 measurements: Left, Top, Width, Height.

Associated Variable. Most controls can be associated with a variable, but it's not required that you do so.

! Bitmap	Bitmap Control
! Push Button	Push Button Control
! Radio Buttons	Radio Button Control
! Check Boxes	Check Box Control
Not covered in this manual	Color Wheel Control
! Combo (Combination) Box	Combo Box Control
Not covered in this manual	Counter Control
Not covered in this manual	Custom Box Control
! Date Box	Date Box Control
! Edit Box	Edit Box Control
Not covered in this manual	File Name Control
Not covered in this manual	Frame Control
Not covered in this manual	Group Box Control
Not covered in this manual	Horizontal Line Control
Not covered in this manual	Vertical Line Control
! List Box	List Box Control
Not covered in this manual	Progress Indicator Control
Not covered in this manual	Horizontal Scrollbar Control
Not covered in this manual	Vertical Scrollbar Control
! Static Text Box	Static Text Control
! File Viewer	File Viewer Control

! Control Names. Each control you insert in a dialog has a case-sensitive name. When a control is inserted, a name is created automatically, such as "Static1" or "ListBx2" or "ComboBx3." But, you will likely want to change control names of controls you need to identify and use in a Callback routine associated with the dialog when the dialog is displayed. Consider changing the names of Static text boxes that may change in a dialog to "S1" or "S2", or interactive boxes of any type to "B1", "B2" or

something else that is easy to identify and type as you enter your other code in the macro. All control names are case sensitive when used in other parts of the macro. While you can use the same name for multiple control names, doing so will probably cause a Callback routine associated with the dialog to fail or work incorrectly. So, give each control a different name. In this regard, see [Chapter 8's discussion](#).

! Control size and placement. Every control has Left, Top, Width and Height placement settings. Select a control by clicking on it in a dialog, then resize it by dragging on its borders or corners; or, double-click on the control to open its Properties dialog. A control's settings are relative to the dialog's dimensional settings and the values are "dialog units." Until reading J. Dan Broadhead's remarks, below, I had no idea what a "dialog unit" was, other than it being some measurement relative to the dimensions of the dialog. He notes that:

A "dialog unit" is a form of measurement used by Windows in dialogs. They are expressed in relationship to the font being used.

From the Microsoft Developers Kit, it says "A unit of horizontal or vertical distance within a dialog box. A horizontal dialog unit is the average width of the current dialog box font divided by 4. A vertical dialog unit is the average height of the current dialog-box font divided by 8."

Suppose that you have a dialog that is 200 wide and 96 tall. This means that the dialog should be wide enough to display 25 characters, and tall enough to display 12 rows of characters ($200/4 = 25$, and $96/8 = 12$).

But that is in the current font, right? What if I wanted to use a 12 point font instead of an 8 point font? Then how big would the dialog need to be to still display 25 x 12 characters? The answer is still 200 x 96. This is because these units are based on the size of the current font. So this means that you can specify the size of a dialog in dialog units, and if you make the font bigger, then the dialog gets bigger too so that the same amount of text could still be displayed.

So now that you understand a dialog unit, you can briefly explain that they are units that are based on the current font size: An average character in the current font is 4 dialog units wide, and 8 dialog units tall.

Consider yourselves "briefly explained"!

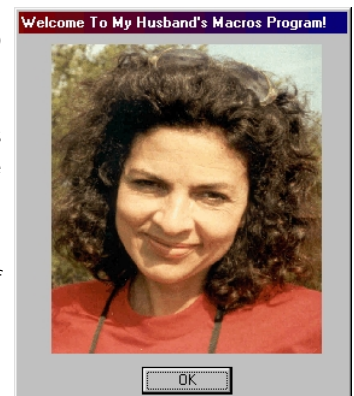
Generally, you will not want a control's Left or Top settings to be less than 2. A "2" value is slightly inside the respective left or top edge, which is zero (0). Width and Height set the size of the control within the dialog.

! Associated Variable. All controls (other than Push Buttons) can be associated with a variable which sets a predefined value to insert into the particular control. If a dialog contains a Callback function, it's best that the associated variable's name NOT be the same as the name of a variable used/obtained in a Callback routine, such as in a `var=RegionGet...` type of command, particularly if you are using WordPerfect 7.0. Unlike control names, variable names are not case sensitive.

SPECIFIC CONTROLS. Aside from the above commonalities, different controls have varying properties. Some, not all, controls are discussed below.

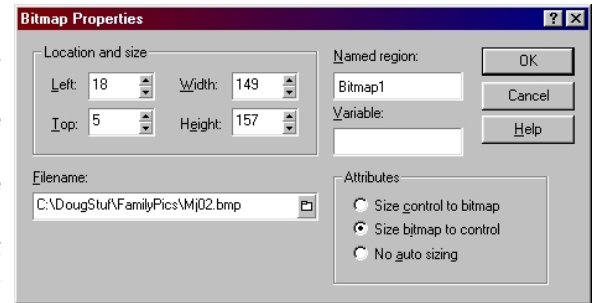
! BITMAP CONTROL. If you want to have some fun and add some pizzazz to a macro, consider adding a Bitmap control, as shown here. The control can only use Windows ".bmp" files (I find no help about this in WordPerfect help, but I've tried .jpg, .gif and .tif files and they won't work). So, if the graphics file is not a *.bmp file, open the graphic file in your graphics software (at the least, you should have Windows Paint on your computer), and save it as a *.bmp file.

Ain't she pretty? And, yeah, I'm the lucky husband, and, yeah, I'm one hell of a luckier guy that I deserve to be! Not a problem!



After adding a Bitmap control (or when editing an existing Bitmap control), double-click on the control to open the Bitmap Properties dialog.

The filename identified in the Bitmap Properties dialog needs to contain the full path and, of course, the file needs to exist before the image file could be used in the macro. I've not figured out the use of the "Variable" which can be identified in the Properties dialog – I've tried assigning a bitmap file to a variable, and not use a filename in the Filename box, but the image does not appear. However, see J. Dan Broadhead's comments, below, for a different approach which may work for you. Ordinarily, Size bitmap to control will be the Attribute choice to make, but try the others out and see what they do.



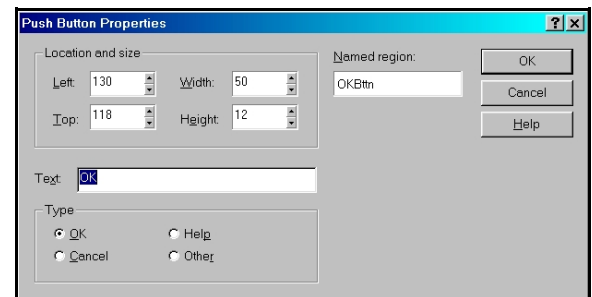
J. Dan Broadhead comments as follows:

For some reason, the "variable" property for a bitmap requires a bit of extra code to get to work properly. I don't know if this is the same across versions, but I got it to work in 9 by doing this: Leave the Filename field for the bitmap property blank. Fill in the variable field with the name of a variable. Just before the DialogShow() command to display the dialog, assign the name of a bitmap file to the variable associated with the bitmap control. Then do a DialogLoad() command for the dialog, and then do a DialogShow() command for the dialog. Now the bitmap file whose name is in the variable associated with the bitmap control will show up when the dialog is displayed.

! **PUSH BUTTON.** Here, the OK button has been selected and its Properties dialog has been opened.



Notice the possible Types of buttons. While a control's name is case sensitive, you can enter the text that will appear in the button as you prefer, e.g., an "OKBtn" doesn't have to read, "OK." It could be "Proceed" or "Yes" or whatever you want. For the OK and Cancel buttons, it's probably best to leave the control's name to WordPerfect's default, "OKBtn" and "CancelBtn."



I've never used the "Help" type button and, after reading J. Dan Broadhead's comments about this feature, I'm certain that I never will:

You must first build a Windows help (.hlp) file using a third party help builder tool. This is the old Windows help format, not the new .chm help file format. You must define a help topic in the help file as well.

Then you must create a help button on the dialog with the HelpBtn! style, and then you must call the DialogSetProperties command, and specify to the HelpFile and HelpString parameters, the name of the help file on the disk and the name of the help topic in the help file (or create the button and specify the help file in the Dialog Editor).

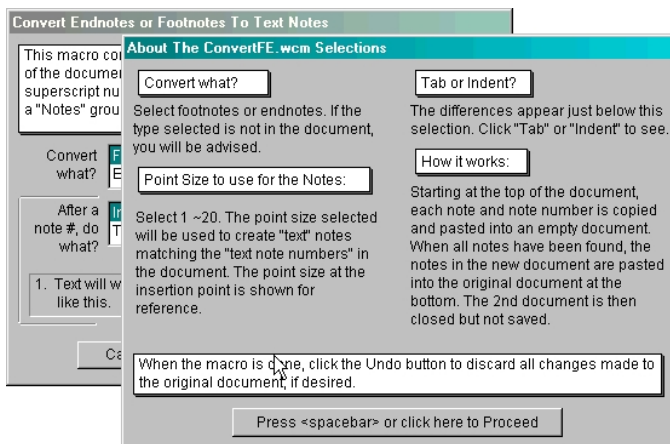
Creating the help file is a mess, since Microsoft did not provide any tools to do that, and you had to go to 3rd party companies for such tools. Then specifying the help topic string is confusing too, and I don't know the proper format for the help topic string.

In my opinion, the Help button is way too complex. You can implement something very similar yourself. When you click an "official" help button, the macro system loads the help file for you, and jumps to the proper help page for that dialog. But if you put a "normal" button on the dialog, which has the text "Help" on the button, then the user can't tell the difference between that and an "official" help button.

Then you could respond to a click on "your" help button, and display either another dialog with the help text, or something similar – maybe even use AppExecute to run WinHelp.exe (the old Windows help viewer), and pass it the name of a help file for WP or something.

You could respond to your help button either in a callback as the dialog is still displayed, or in a non-callback dialog after the dialog is closed by clicking on the help button (after which you would probably want to redisplay the original dialog).

Aside from the "Help" button type, I frequently use "Other" buttons to open "help" dialogs, naming the control "TipsBtn", or "HelpBtn" and the like, which are used to open other dialogs which give contextual help associated with the main dialog, and without closing the main dialog which is opened during a callback routine. For example, in the dialog being used to illustrate this discussion, note that **the dialog** contains a "Tips" button. So, when that button is clicked while the dialog is running, the main dialog remains present while the particular "tip" dialog opens. This approach can only be used in dialogs using Callback routines. In this example, the secondary dialog appears on top of the dialog containing the "Tips" button, as shown below.



Here, the "Tips" button has been clicked. The control name I've used is "TipsBtn" but you could use any name you want.

In the Callback routine, when that button is clicked, I previously used DialogShow ("vTips"; "WordPerfect") command to show the "help" or "tips" file – *but being properly educated by J. Dan Broadhead in his comment below*, I now use DialogShow("vTips"; "vType"), where "vType" is the name of the parent dialog.

The "Press <spacebar> or click here to Proceed" button is really an "OKBtn" in disguise. Clicking that button or pressing <spacebar> or <Enter> closes the dialog and focus is returned to the main dialog, shown in the background in this picture.

As I've said, I prefer this approach to using **Message Boxes**, explained in Chapter 10.

When ConvertFE.wcm was originally written, using "WordPerfect" as the 2nd parameter in DialogShow was the only way that I knew (as in DialogShow("vTips"; "WordPerfect")). But, J. Dan Broadhead teaches a better way (and this is particularly important in WordPerfect 8, which is more prone to lose a running macro's focus – i.e., it can get "hidden" and you won't even know it's there):

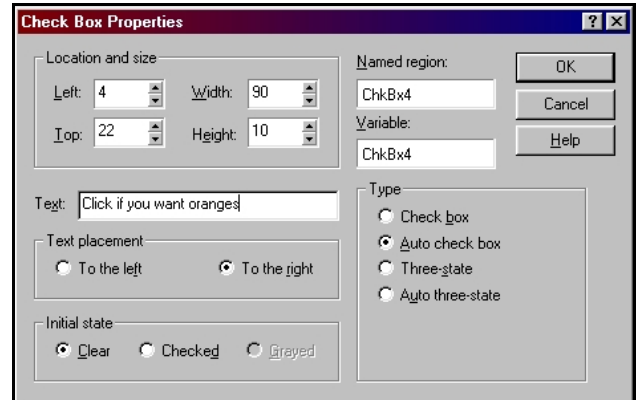
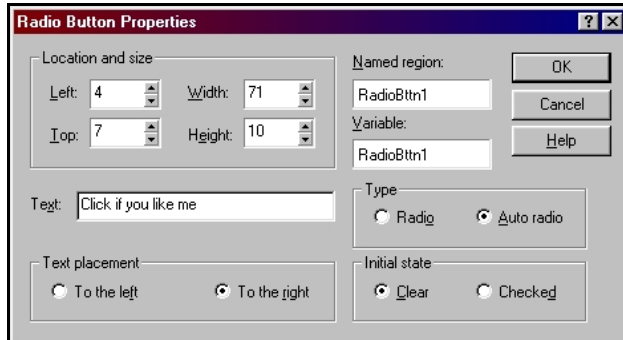
You use DialogShow ("vTips"; "WordPerfect") to display your "tips" dialog. I would suggest that you change the second parameter from "WordPerfect" to the name of the main dialog. This second parameter is the parent window for the new dialog. By setting it to "WordPerfect", then both the main dialog and the "tips" dialog are sibling dialogs of the same parent, and the "tips" dialog could possibly fall behind the main dialog in some circumstances. Since the tips dialog was displayed from the main dialog, then in Windows, the tips dialog would normally be considered as a child of the main dialog, and so you should specify the name of the main dialog as the parent for the tips dialog. This will make sure that the proper focus and layering issues are preserved.

Also, see **SetDefaultParent**, generally. I surely would like to have even known that such a thing was possible many years earlier! As far as I knew, "WordPerfect " was the required second parameter of DialogShow! We live and we learn!

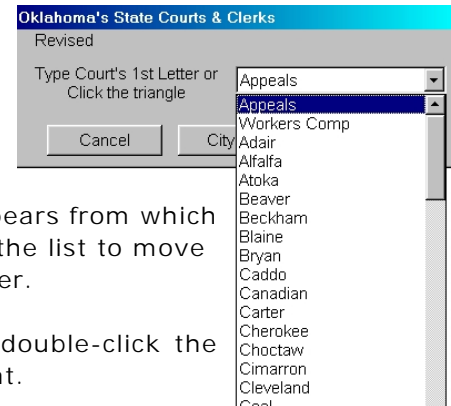
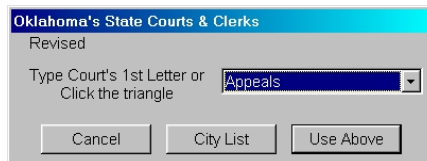
I'm kind of straying off topic, so let's get back to the controls.

! RADIO BUTTONS AND CHECK BOXES. While similar, they're different. Radio buttons (at the top of this picture) usually provide alternatives, one or another. Check boxes can be all or none. As set up here, clicking the left Radio button selects or deselects the right Radio button and vice versa. Check boxes and Radio buttons return a value of 1 or 0 – zero if unchecked, 1 if checked. Either has various properties, none of which are further explained in this manual, but read on-line Macro help for more information.

As always, double-click on the control to open the control's Properties dialog. The properties of each control shown in the above picture are shown below.

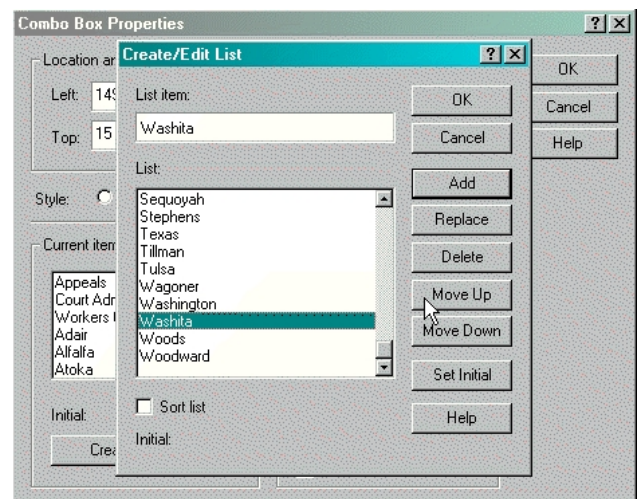
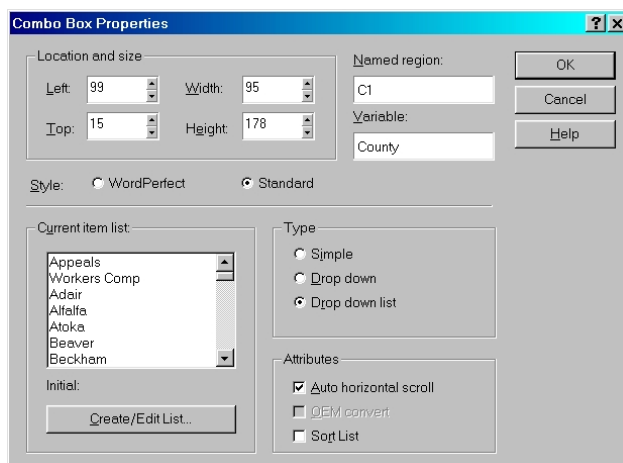


! COMBINATION BOXES. A dialog using a "Combo" box is shown here. The Combo Box Properties dialog below shows the Properties for this particular Combo Box. When the macro is running, if you click the triangle at the end of a combo box, a drop down/up list appears from which a selection can be made. A user can type the 1st letter of an item in the list to move to it by rotating among selections in the list that begin with that letter.



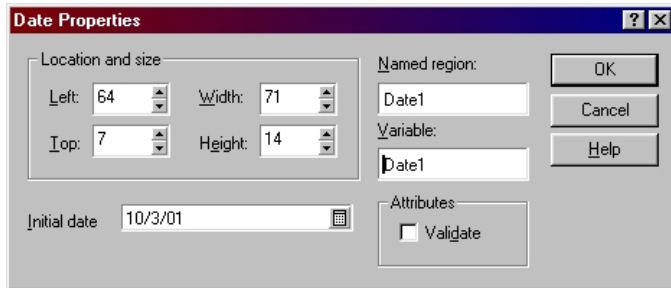
In the Dialog Editor with the dialog you intend to work on open, double-click the Combo Box to open its Properties dialog. There, select what you want.

In the Combo Box Properties dialog, click the Create/Edit List ... button. That opens the Create/Edit List dialog. I'll not describe this closely the dialog is quite intuitive. Select one item to be the default and click the Set Initial button. When done, click the OK button in the Create/Edit List dialog to



return to the Combo Box Properties dialog. A list can be "Simple", "Drop down", or "Drop down list." If "Simple", the box is open, showing items in the list – probably not what you want. If "Drop down", a user can "type-in" something that's not in the list. If "Drop down list", choices are limited to items in the list. When done, click the OK button to return to the dialog itself in the Dialog Editor.

! **DATE BOXES.** As shown in this simple dialog, a Date box control can be used to obtain a date. If clicked, the calendar icon at the end of the box pops up a calendar for a user's use, as shown in the lower right picture. Single triangles move a month at a time, and double triangles move a year at a time.

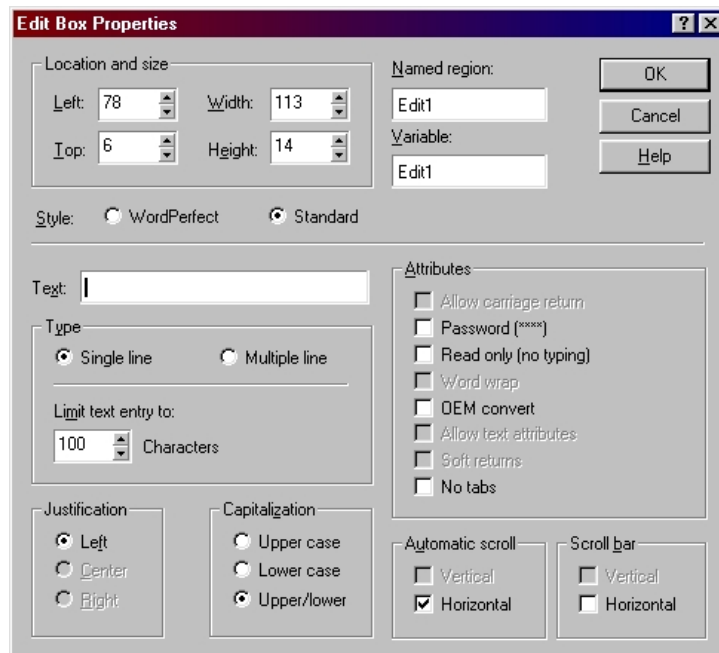
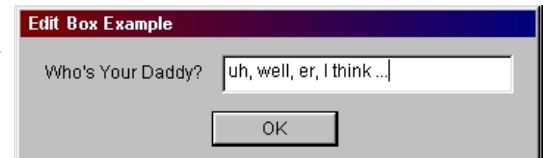


Before WordPerfect 8 Service Pack 7, dates default to two digit year dates which can create Y2K problems (left picture). After that, a 4 digit year date should be the default.



Instead of Date Box controls, I've come to use Edit Box controls which force an mm/dd/yyyy date format. Date Boxes don't always work well for cross-WordPerfect version macros (6.1, 7.0, 8.0 and higher, but consistent in WordPerfect 8.0 SP7 and higher). For these reasons, I avoid Date Boxes. But, if cross-WordPerfect version compatibility issues aren't important to you, Date Boxes can be handy.

! **EDIT BOXES.** These boxes are used to obtain a user's text response – the user enters any text desired. Upper or lower case can be forced if desired. As always, double-click on the control to open the control's properties dialog, in this case, the Edit Box Properties dialog, shown below.



Set the Edit Box's properties here.

The box can be single or multiple lines; input is limited to a specified keystroke number; and case can be forced.

So, to force a state postal abbreviation, limit the characters to 2 and force upper case.

You can use it to obtain a numeric response, but the value will initially be a string value. To change it to a numeric value, use StrNum(var) or other options. See [Chapter 6](#). If the content of the initial string contains non-numeric characters, an error will occur if you use StrNum(var) to test the value. So, if you intend that a user only enter string values which would convert properly to numeric values (+-0123456789 and "period" (decimal)), be sure to include **error checking routines** such as that shown in [Chapter 6](#), below

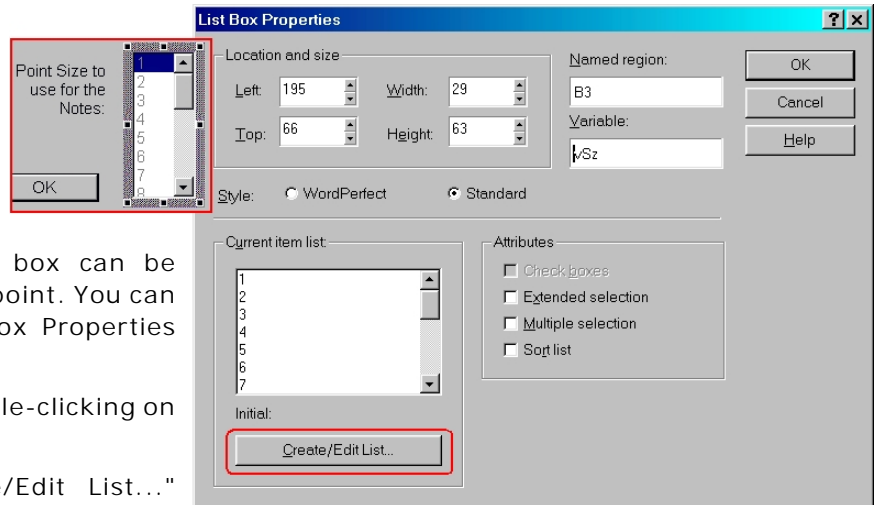
See on-line Macro help for information about the other options.

! **LIST BOXES.** A list box contains a list of string-value choices. These images are a cutout of a selected list in the picture at the left and the Properties dialog for that list.

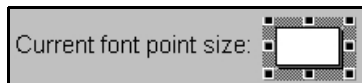
In the left picture, notice the selection points around the box. The vertical and horizontal size of the box can be resized by dragging on any selection point. You can also resize the control in the List Box Properties dialog.

The List Box Properties opens by double-clicking on the List Box control.

In this dialog, notice the "Create/Edit List..." button. Click that button to open the list itself for editing. There, in addition to setting/changing values in the list, you can set a "default" initial value, which you should do. "Attributes" include "Extended" and "Multiple" selection, as well as "Sort List." Extended and Multiple allow multiple selections to occur. Extended allows a user to click an item, press shift key and select another item which selects all items in between. Multiple allows a user to select/unselect an item by clicking on it.

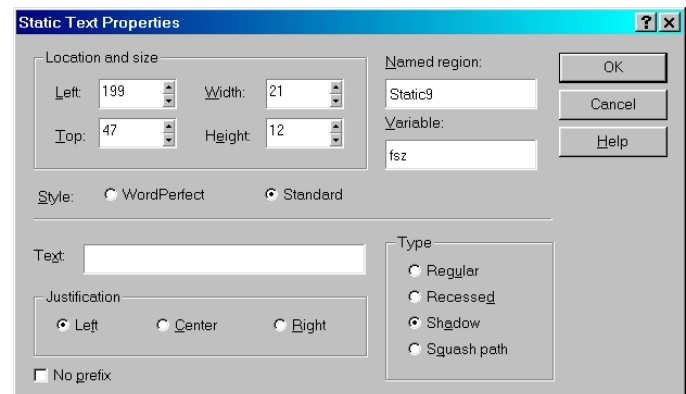


! **STATIC TEXT BOXES.** It contains informative text so that a user will know something. The cutout picture, below, shows a Static Text box which has been selected – the one after the "Current font point size:" Static Text box. As always, double-click on the control to open its Properties Dialog.



In this example, no default text is used for the "Text" edit box. Instead, a

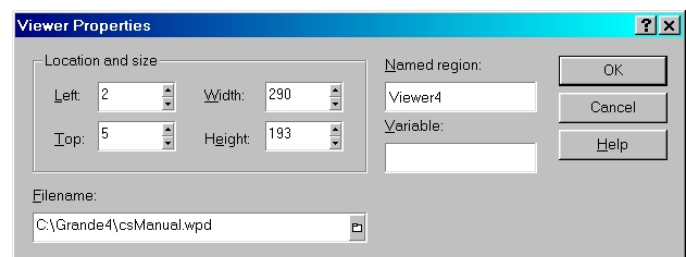
variable named "fsz" is associated with the box. In a routine before the dialog is displayed, the current font point size has been assigned the variable "fsz" so that the user will have that information in making a selection for the size of the notes to be made with the macro itself. Note that you can set Justification as Left, Center or Right, and that 4 Types of display can be used – Regular (which is used in the "Current font point size" main dialog text box), Recessed, Shadow (used in the above) and Squash path. Experiment to see the differences.



! **FILE VIEWER BOXES.** Given the matters described here, the use of Viewer Boxes can present problems and isn't worthwhile using in WordPerfect 11.0 or later, in my opinion. Read this section thoroughly before using Viewer Box controls and then do what you want.

! **Viewer Installation, Generally.** In some versions of WordPerfect (and I don't recall which – possibly 6.1 through 10.0), viewers may not have automatically installed or perhaps, in a custom WordPerfect installation, a user opted not to install viewers. In either event, a macro which contains a Viewer Box control may lock the macro and WordPerfect. If you intend that macros be used in any WordPerfect installation, it's best not to use Viewer Boxes in your macros.

! **Use of the Control, Generally.** Create a Viewer control as you would with any other control – click the icon representing the control (eyeglasses) in the Dialog Editor toolbox and then click where you want in the dialog you are editing. Resize it as you want. When ready, double-click on the control to open the Viewer Properties dialog.



Identify a file to be used in the Viewer Box. Either type-in a file's full pathname, or click on the folder icon at the end of the Filename box to browse for the file. A "Viewer box" allows you to identify a file which will be "viewed" in a WordPerfect dialog containing a "Viewer box" component. The file can be a WordPerfect document file, an ASCII/ANSI (".*txt") file, a Rich Text Format (".*rtf") file or other file formats supported by the Viewer software (if any). If you're not satisfied with the control's placement in the dialog, set what you want for Left, Top, Width or Height. I'm not clear on the use of a Variable in this context and I haven't used that feature.

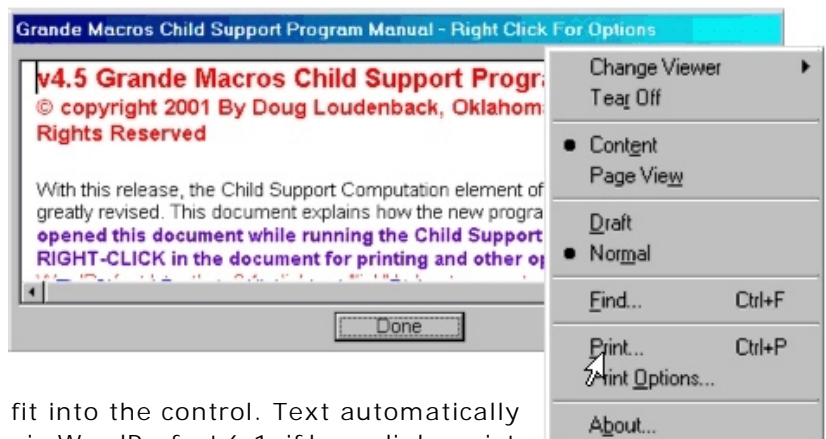
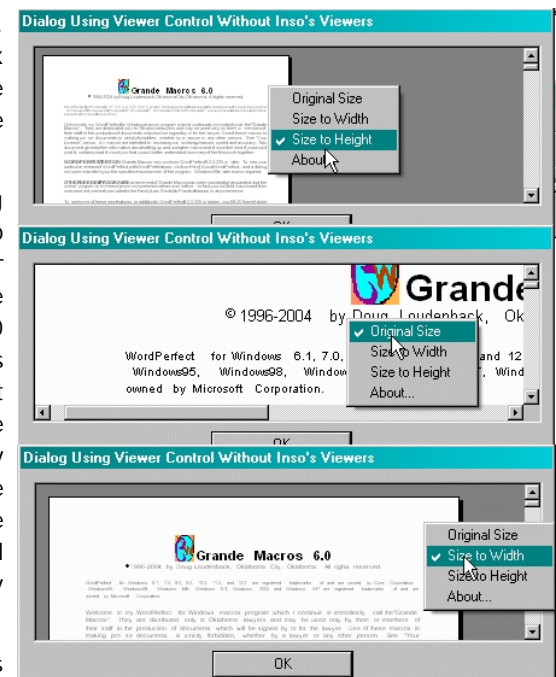
! Viewer Boxes Made in Different WordPerfect Versions. Macros made in WordPerfect 6.1 which contain Viewer Box controls will not work in WordPerfect 7.0 and vice versa. Macros made in WordPerfect 8.0 and higher which contain Viewer Box controls will not work in WordPerfect 6.1 or 7.0. Assuming that viewers have been installed, macros made in either WordPerfect 6.1 or 7.0 which are run in WordPerfect 8.0 or later will work. Other than the absence of Inso Corporation's Outside-In viewers, discussed below, no Viewer Box control issues are present in WordPerfect 8.0 and higher.

! Viewer Boxes Using Native WordPerfect Viewers. These three pictures show a dialog containing a Viewer Box control using WordPerfect 11's built-in viewer. Each picture shows the pop-up box presented with a right-click in the Viewer Box: Size to Height, Original Size, and Size to Width.

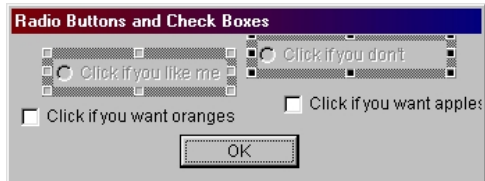
Beginning with WordPerfect 11, Corel stopped including special add-on viewers formerly supplied by Inso Corporation, described below, which viewers made Viewer Boxes so very useful in pre-WordPerfect 11.0 versions. While Viewer Boxes are still available in WordPerfect 11.0 and 12.0 as shown here, as a practical matter the text presented is either too large or too small, it won't resize properly to fit within the control so that it's easily readable, hyperlinks in the document won't work, and you can't print the document by right-clicking on the control and selecting Print, since the option doesn't exist. Put simply, the options are too few. The reduction of a Viewer Box's capabilities in WordPerfect 11 and 12 render the use of Viewer Boxes rather useless, in my opinion.

! Viewer Boxes With Inso's Outside-In Viewers Installed. *Ahh ... these are great Viewer Boxes!* In WordPerfect 6.1 through 10.0, a WordPerfect installation option, but not selected automatically, is to install Inso Corporation's "Quick View Plus" and its "Outside-In" viewers. For some WordPerfect versions, that's on Installation Disk 2, for others it's available on Installation Disk 1.

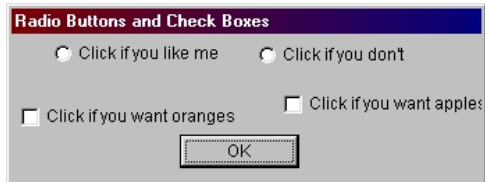
This picture shows an example of a dialog containing a Viewer Box control with Inso's Outside-In viewers installed and the benefits: Automatic Word Wrap – the document is automatically resized to fit into the control. Text automatically wraps so that it reads easily. Hyperlinks – in WordPerfect 6.1, if hyperlinks exist in the WordPerfect document used in the Viewer Box control they won't work – but they will in WordPerfect 7.0 through 10.0 IF the Inso Outside-In viewers are installed. Hypertexted WordPerfect documents are especially useful for long documents when used in a Viewer Box control, e.g., a research paper or a manual of some type. Print and Other Options – a user can print the file contained in the Viewer Box (Ctrl+P), can find text located in the file (Ctrl+F), as well as perform the other options shown. All this is gone in WordPerfect 11 and higher.



! ALIGNING STUFF IN THE DIALOG. Though you can use the Grid controls in the Editor Toolbar, I never do. Instead, I select multiple items to align, right-click and select what I want. Either way, the last item you select will be the benchmark, so to speak, for any other selections you've made. I've messed up the vertical and horizontal alignment here. Let's make it right!



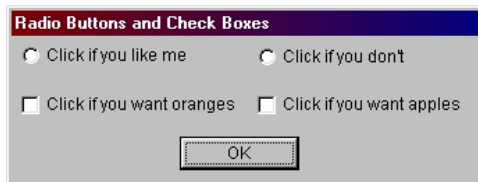
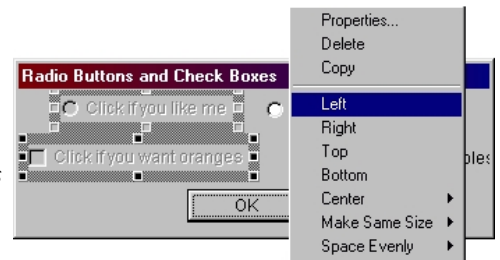
First, let's correct the horizontal alignment of the radio buttons. Click on the one that's badly aligned, and *then Shift+Click on the one that's right*. Both will then be selected as shown here.



Second, right-click on either selected item to get the pop-up menu superimposed over the dialog. Since, in this case, I want to align the selected items to the "Top" of the last item selected, I'll select "Top" in the pop-up list. That's all there is to it, and the items selected are now aligned where I wanted them to be.

Without showing it, I've done the same with the check box items, thus making the horizontal alignment the way I want.

Still, the vertical alignment needs fixing. I'll select the item that's out of line (the top radio button), *then Shift+Click on the one that's right* (the checkbox), right-click on either and get the same pop-up list. This time, I'll select "Left." Now they are aligned at the left.



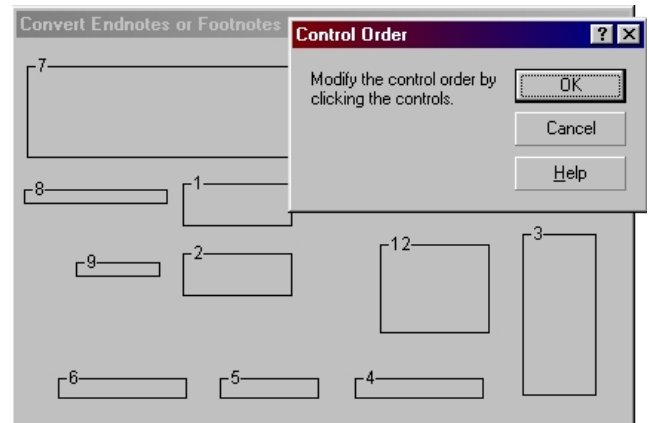
Without showing it, I've done the same thing to the other pair, on the right side of the dialog. Now, all is as it should be.

In the same way, you can use the other items in the pop-up menu, Make Same Size (vertical, horizontal, both), Center (vertical, horizontal), Space Evenly (vertical, horizontal), Right, Bottom.

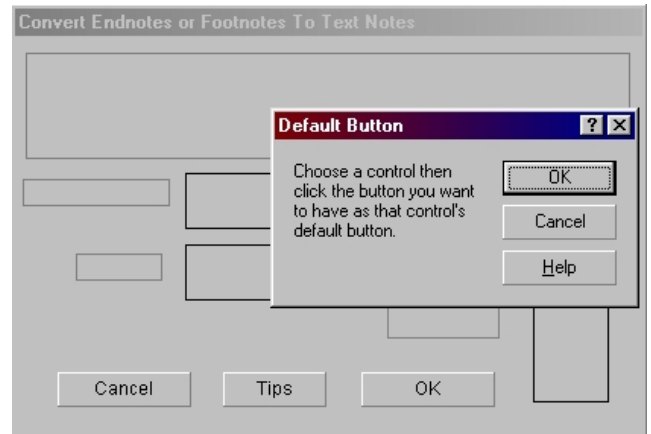
After you've done this a few times, it will come easy. And, note that you can select multiple items (using Shift+Click) and *drag all items simultaneously* to where you want them to be.

• ORDERING THE CONTROLS.

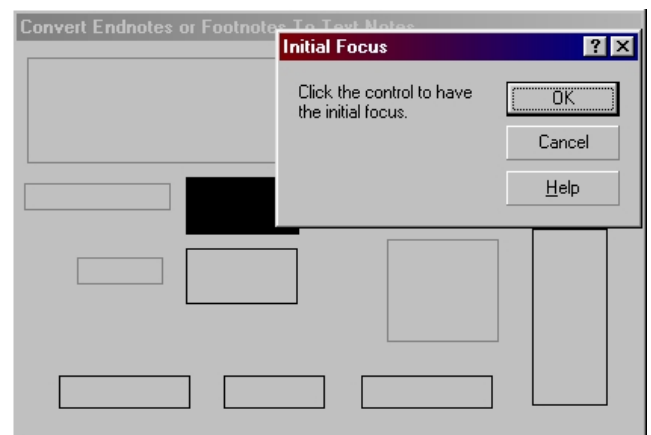
! Controlling the Sequence. In the dialog, and with nothing selected, right-click to get the pop-up list described before. Then, select Control Order. The pictures shown here appear. If it's in the way, drag the Control Order box out of the way. Then select the 1st control in the sequence you want, then the 2nd, etc., until all in the sequence are identified. You don't need to set the order for Static Text boxes – they are skipped anyway. Only boxes, buttons, etc., which receive a user's input need to be ordered. When you are done, click the OK button in the Control Order box.



! Default Button. If you have more than one Push Button, it's best that you select the one you want to execute by default if the Enter key is pressed (sometimes, the Spacebar key will do the same). As before, with nothing selected, right-click in the dialog and choose Default Button from the list. Then, click the Push Button to have precedence and click the OK button in the Default Button box.



! Initial Focus. Regardless of how you may have set the order of controls, you can and should set a control to have the initial focus when the dialog first opens. As before, with nothing selected, right-click in the dialog and choose the control to have the initial focus. Then, click the OK button in the Initial Focus box.

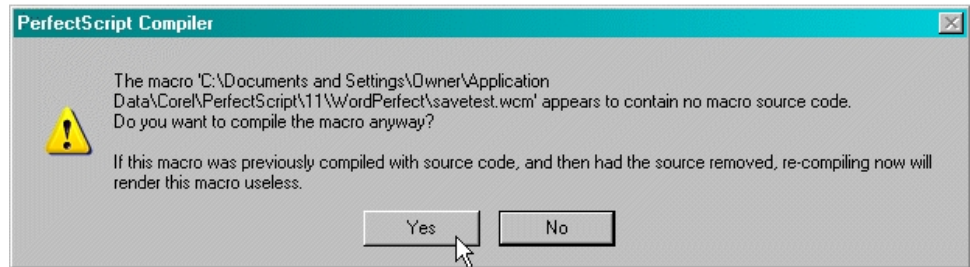


! When Done, Save Changes. Naturally, you need to close the Macro Editor. You can click any of the 3 buttons highlighted here -- the 2 on the left (save, close) or the Close "x" button in the upper right corner of the toolbar. If you chose either "x", you will be asked if you want to save any changes you have made. "Yes" saves the changes, "No" doesn't and the dialog will be as it was before you started your editing.



That's the end of my discussion about use of the Dialog Editor. Of course, there's more that could be said, but, hey, this is "A Common Person's" macro manual, and what I've said is enough about that!

- **ENTER MACRO CODE AS NEEDED.** The Dialog Editor will not be available until at least an attempt at saving the macro once in it's existence has been made. For example, below, I've done the following using WordPerfect 11: (1) Opened a totally blank document; (2) Turned on the **Macro Toolbar** by using Tools | Macros | Macro Toolbar, and (3), with nothing at all in the document, clicked the Save & Compile button on the Macro Toolbar. This message appeared:



Whether I clicked the Yes or No button, the Dialog Editor button was then available for use on the Macro Toolbar and it was possible to create dialogs within it. Whether this is so for all WordPerfect versions, I don't know. Of course, as a practical matter, you'd probably not be writing a macro **totally from scratch**, anyway, but would have at least **created a shell**, both described in Chapter 3.

As qualified above, the order in which you edit a macro file, i.e., whether you first edit dialogs, as shown above, or first edit macro code in the actual macro file document and then (if needed) edit dialogs in the macro, is not important. This section doesn't develop "how" to enter macro code from the keyboard – that varies greatly depending on what you want. But, generally, type in the code that matches what you want the macro to do, using the syntax described in **Chapter 10**, and see examples of macro code to be entered/ edited in various chapters in this manual.

- **CLICK THE SAVE & COMPILE BUTTON.**



After making changes to the macro file, you must Save & Compile the Macro to

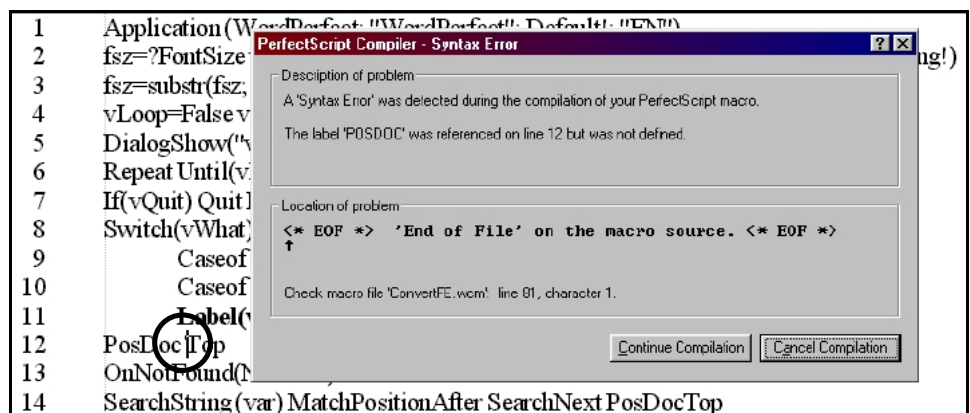
save any changes you have made. That's easy – click the Save & Compile button on the Macro Toolbar, as shown above. After doing so, the macro is compiled. When that is done, the Save & Compile button on the Macro Toolbar will be "dimmed" and the "File New" icon on the regular Icon Toolbar will be "white", not dimmed. During compilation, a message, "Macro compiling" or "Compiling macro" will be on the WordPerfect "Taskbar" – in Wp6.1 and 7.0 it's so dim you may hardly notice it – in Wp8.0 and higher, it's a bit more noticeable.

Also, during a macro's compilation in Wp8.0 and higher, an icon will be in the Windows system tray which looks like 3 horizontal pieces of paper laying on top of each other.

Time needed to compile macros varies depending on your computer's memory and the complexity and length of the macro. Usually, it takes is a split second, or just a few, but some very complex macros may take longer than a minute to compile.

- **ERRORS AND WARNINGS DURING COMPILATION.** Unless you were Mr. or Ms. Perfect, errors and/or warnings will be reported during compilation telling you a little bit about why you were bad! Hordes of error or warning messages are possible, and only a few are considered here, just to show you what to expect.

Here, on Line 12, I've deliberately entered a space between "PosDoc" and "Top." The correct command to position the insertion point at the top of the document, after codes, is "PosDocTop", no spaces. The "PosDoc" statement was reported as a "Label" – which it isn't, but the Line 12 was identified as was the particular string containing



the error. Line identification isn't always exact, particularly in WordPerfect 6.1, but it should get you pretty close. In the above example, even though the "Location of problem" part of the message, "End of File", isn't helpful, the identification of "line 12" in "Description of problem" is (since I didn't intend to make a label at Line 12 but merely left out an important space).

Occasionally, a harmless *Warning* (as opposed to *Error*) will be reported, such as that a Label(xx) was defined but not referenced (even though being referenced *but not defined* is quite another matter). You can ignore "defined but not referenced" *warnings* and click the Continue Compilation button. The macro will compile. Maybe you just weren't finished writing the macro and wanted to make sure what you'd already done was OK. But, if you receive an *Error* and not a *Warning*, the macro will not successfully compile until you have fixed whatever is broken.

Here's a biggie Error: Statements having a required beginning and a required ending are the hardest of all to find since an End of File location will be identified as the point of error. For example, every "If()" statement must end with an "EndIf" command. If you have pages and pages of code and lots of "If()" statements, finding the broken point can wear you out – the absence of an EndIf will be reported as being at the end of the file since, technically, that's

where the error was found – the EndIf statement was never found and, since it could theoretically occur anywhere in the macro document, and was never found through the end of the document, the compilation warning will not be able to tell you where to look for the error.

In the above picture, notice that I've "commented out" the EndIf at Line 7 – comments are ignored during macro compilation.

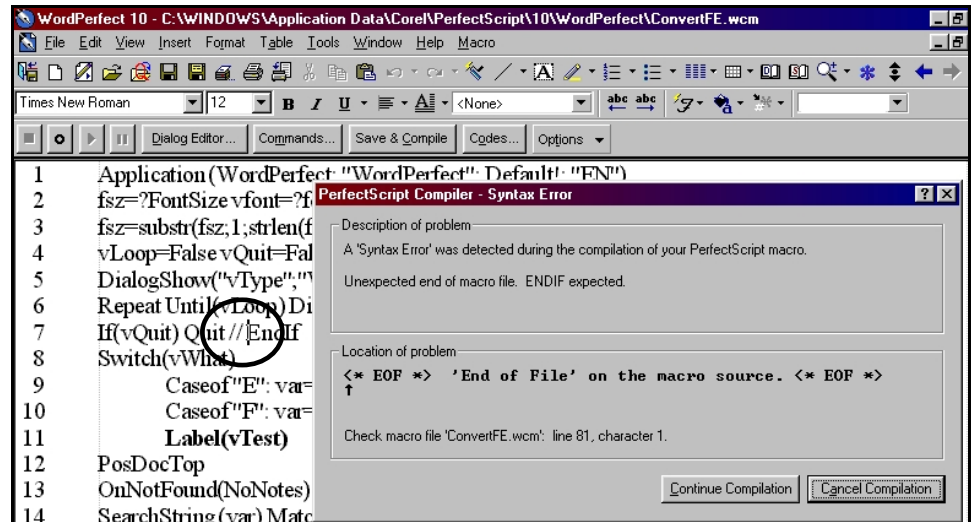
So, in this example, the Line reference is not helpful – Line 81 is the last line of macro code. That's the type of report generated in the PerfectScript Compiler Syntax Error dialog when an EndIf is missing. If you've got a lot of If statements, particularly if you have a lot of If-within-If statements, finding the place that the EndIf should have occurred can be rather tedious in a very long macro.

Teaching point: Be very careful as you enter code to include all of the elements necessary to write a successful macro, particularly for *statements requiring a pair of codes*, beginning and ending (**If – EndIf**; **While – EndWhile**; **Prompt – EndPrompt**; **Switch – EndSwitch**; **Functon – EndFunc**; **Procedure – EndProc**).

Also, take heed of the following comments by J. Dan Broadhead concerning errors:

It is fairly common for one error to trigger a whole series of other mysterious errors on what might appear to be valid code. When an error occurs and you choose to "Continue" the compilation, the compiler tries to skip over the error, and re-synchronize itself with your macro code. Doing so may cause it to skip ahead a bit, and thus not detect some essential code which could cause other valid statements to appear to be in error.

When correcting multiple errors, it is best to correct the obvious ones that you understand, and skip the more obscure one because they may be bogus due to some prior error, and may go away when a prior error has been corrected.



Indentation practices when writing macros can help avoid missing "EndIf"s, or to locate where they ought to be. He writes:

I have a suggestion that comes from years of dealing with these same mismatched commands. It has to do with structuring the source code to make reading it easier, especially when it comes to matching up the begin and end paired statements.

Programmers have found that indentation and statement placement in the source can significantly improve one's ability to comprehend the code, and to locate mismatched errors. But the actual techniques used can vary greatly.

Here is what I have found that helps me.

The statements that have matched begin/end pairs are called "compound" statements, because they begin at one point and end at another point, and can contain other statements within their begin and end boundaries. Lets take the If/Else/EndIf compound statement "pair".

I always place the begin and end statements for the compound statements on separate lines by themselves, and they are indented to the same level as each other. I then indent the statements within the "body" of the compound statement by 1 tab stop (I usually set the tab stops to be on 1/2 inch marks), like this:

```
If (something)
    do something do something else and something else
Else
    do the other stuff
Endif
```

If I have another compound statement within the body, then its own body is indented one level deeper:

```
If (something)
    do something do something else
    If (some other thing)
        do the other thing
    Endif
    and something else
Else
    do the other stuff
Endif
```

Now I find it easy to see if I am missing the EndIf.

I find the If, and then look down at the next statement that is indented to the same level, and it should be the EndIf (or the Else followed later by an EndIf). All the code that is indented deeper than the If statement itself, is part of the body of the If.

It also helps a great deal if only a single command is placed on a line, though PerfectScript will support more.

Generally, those just starting to write macros won't practice any indentation at all, and it becomes quite difficult to locate the missing commands.

This also makes it a lot easier for other people to follow the code as well, since they are not familiar with the motivation behind the code in the first place, and clear indentation practices helps make the code clearer to interpret.

NOTES

Chapter 6

Math Routines

Links: The **Chapter Name**, above, moves to the next chapter. 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.

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				
Math Operators	General Rules	Automatic Data Conversion	Addition	
Subtraction	Multiplication	Division	Floating Cell Problem	
Massaging Numbers	Working with Integers	Working with Decimals	Specific Tasks	
Commas	Making it look right	Percentages	Error Trapping	

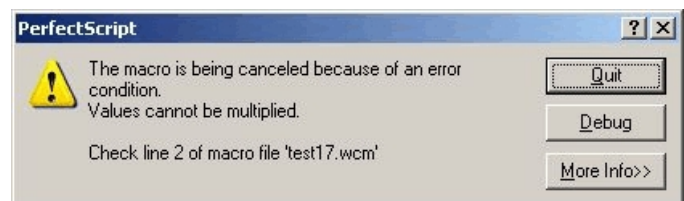
This chapter covers a few of the many math routines possible in WordPerfect macros. It gives some basics but it is by no means comprehensive. For example, it does not discuss the concepts of "expressions" or "operator precedence" at all, except to say that, for purposes of this manual, an "expression" is a statement containing an assignment of some value to something, usually a variable (which I tend to call a "formula"), e.g., `x=1` or `var3=var1+var2`. Concerning expressions, see <http://jdan.com/perfectscript/macros/ch04.htm>; for operator precedence, see http://jdan.com/perfectscript/macros/ch04.htm#_VPID_04_24. Even so, this chapter should give you a good working knowledge of math operations you'll likely need to know.

! MATH OPERATORS. `*` is used for multiplication; `/` is used for division; `-` is used for subtraction of numbers and reduction of strings; `+` is used for addition of numbers and concatenation of strings; `%` is used for floating point modulus division and returns the floating point division remainder; `MOD` is used for integer division and returns the integer division remainder; `DIV` is used for integer division and returns the integer portion; and `**` is used for exponentiation or raising a number to a power. Only `*`, `/`, `+`, and `-` are discussed in this manual. Note that both `+` and `-` are string operators, as well.

! GENERAL RULES. Math routines operate upon "numeric" (number), not "string" (text) values. Subject to what's said in **Automatic Data Conversion**, below, when using math it's extremely important that the values you are using numeric, not string, values. Otherwise, error or an unintended result may occur and, if so, in the absence of some error trapping routine, the macro will fail or work differently than intended. An error dialog similar to the picture below most probably means that one or more of the values or variables containing values are not numeric.

If `var1="$1"` and `var2=1`, `var1` contains a string value which cannot convert to a numeric value whereas `var2` contains a numeric value.

While WordPerfect will sometimes "know" that a variable containing a value which could be either a string value or a numeric value (e.g., `var="1"` and `var=1` may be interpreted by PerfectScript as numeric, that's not always so in different WordPerfect versions. So, if you're interested in macros working the same way from WordPerfect 6.1 forward, be sure you are using numeric and not string values when doing math.



Since I routinely want my macros to work cross-WordPerfect versions whenever possible, I tend to use routines which reduce math operations to the lowest common denominator in the various WordPerfect versions from 6.1 to the current version. That predisposition affects some of what I say in this Chapter.

! AUTOMATIC DATA CONVERSION. Probably beginning with WordPerfect 7.0 (it may have begun in WordPerfect 6.1 but I don't think so), PerfectScript treats "strings" which would be the equivalent of numbers (but for the fact that they are strings) as numeric, if called upon to do so under certain circumstances. For example, if `var="1.5"`, `var2=var*2` returns the numeric value of 3.0 to `var2`; and if `var1="1.5"` and `var2=50`, `var3=var2/var1` returns the numeric value of 31.25. Similarly, a numeric value may be treated as a string under certain circumstances. For example, if `var=3` (numeric), `var2="$"+var` returns the string value of "\$2". This section describes what will, or may, or cannot happen concerning automatic data conversion.

- The **+** and **-** operators. These operators aren't just for math (numeric values), they also work with string values, and, if an expression contains multiple values or variables, the combining of values will result in either concatenation (for the **+** operator) or reduction (for the **-** operator). Automatic data conversion isn't involved if all components in an expression are already what is intended (numeric or string), be the intention to work with numeric values (as for math, e.g., `var=1+2`) or string values (as for strings, e.g., `var="1"+"2"`). In the former expression, `var=3` (numeric), but, in the latter, `var="12"` (string).

- String Values. As described in the Glossary (Chapter 10), **concatenation** is the combining of two or more string values, and **reduction** is the (non-math) subtraction of one string value from the value of another. *Reduction became available in WordPerfect 7.0*, and is not available in WordPerfect 6.1.

- Concatenation. If `var1="1,000"`, `var1="$"+var1`, the result is that `var1="$1,000"` (string), or if `var1="1,000"` and `var2=".05"`, `var1=var1+var2`, the result is that `var1="1,000.05"` (string).

- Reduction. If `var1="1,000.05"` (string) and `var2=","`, in `var1=var1-var2` the result is that `var1=1000.05` (`var1` no longer contains the comma). Without more, this technique removes the 1st occurrence of `var2` (the comma) in `var1`, but not any subsequent occurrences. For that, you'd need to use a loop statement of some kind, such as, where `var1="1,000,000,000.00"` (or any other string variable), and `var2=","`:

```
x=StrPos(var1;var2)
If(x>0)
  Repeat
    x=StrPos(var1;var2)
    var1=var1-var2
  Until(x=0)
EndIf
```

- Numeric Values. If `var1=1000` and `var2=.05`, `var1=var1+var2` results in `var1` equaling 1000.05 (numeric). With the same variables and values, `var3=var1-var2` results in `var3` equaling 999.5 (numeric).

- When Automatic Data Conversion Comes Into Play. What's said above used true string or numeric values in each example, so automatic data conversion was never involved. But, if one (or more) values are numeric and one (or more) values are string, then automatic data conversion comes into play.

Initially, PerfectScript assumes that the 1st value encountered in an expression is indicative of the value type (string or numeric) that you intend. For example, if `var1=1` (numeric):

`var3=var1+var2` assumes that a numeric computation is desired, since `var1` is numeric. `var2` will automatically convert to a numeric value, if that is possible. So, if `var2=".05"`, `var1=var1+var2` results in `var1` having a value of 1.05 (numeric) since `var2` can convert to a numeric value via automatic data conversion.

But, if `var2` cannot be converted to a numeric value, e.g., `var2="ABC"`, even though the initial intention was interpreted to be that a numeric value should be attributed to `var2`, since that cannot happen (`var2` cannot convert to a numeric value), a string value results instead in `var3=var1+var2`, and `var3` is assigned the result of "1ABC".

- **Forcing A Presumption.** To force PerfectScript to presume that a numeric expression is intended, you can begin the expression with a numeric value which does not affect the computation. To force a string presumption, you could begin the expression with a pair of quotation marks, e.g., `var3="" + var1 + var2`.

For math addition and subtraction, the expression could begin:

`var3=0+var1+var2` or `var3=0+var1-var2`

For math multiplication or division, use a 1 multiplied by the 1st "real" value:

`var3=1*var1*var2` or `var3=1*var1/var2`

- **Shifting Presumptions.** If you haven't forced a numeric presumption, above, unless all values in an expression are string values, automatic data conversion may nonetheless occur. The presumption may shift at least once, but perhaps many times in long expressions.

- **Two Values.** If only 2 values are involved in an expression, even if the 1st value is a string capable of converting to numeric and the 2nd value is numeric, it is then assumed that a numeric result was intended. So, where `var1="1"` (string) and `var2=0.05` (numeric), in `var1=var1+var2`, the result returned to `var1` is 1.05 (numeric). If the 1st value (string) cannot be converted to a numeric value via automatic data conversion, the 2nd variable will be treated as a string and the expression will return a string value, e.g., where `var1="$"` and `var2=1000`, in the expression `var1=var1+var2`, `var1` will equal the string value "\$1000", meaning, among other things, that the result of the formula/expression is a string, and not a numeric, value.

- **More Than Two Values.** If an expression contains more than two values, it is helpful to understand that:

- **If The 1st Value Is Numeric.** The remaining values or variables will be attempted to be interpreted as numeric. If they are, even if subsequent values are consecutive string values, those values will be attempted to be treated as numeric. So, if `var1=1` and `var2=".05` and `var3=".001"`, in `var1=var1+var2+var3`, `var1`'s value will be 1.051 (numeric).

- **If The 1st Value Is String.** If you haven't forced the numeric assumption (above), if the 1st value is a string, all consecutive string values thereafter will be treated as strings and concatenation will occur for all such values, unless and until a numeric value is encountered. If a numeric value is encountered, the string value which preceded it will be treated as numeric and a numeric result will obtain. So, if `var1="100"`, `var2="5"`, and `var3=8`, the formula `var4=var1+var2+var3` returns a value of 1013 (numeric) to `var4`. In the formula sequence, `var4` 1st became the value of `var1` ("100"); 2nd, concatenation occurred when added to `var2` ("5"), causing `var4` to become "1005"; 3rd, encountering the numeric `var3` (8), "1005" was treated as numeric and `var3` (8) was added to it, resulting in `var4` equaling 1013 (numeric).

- **Examples:**

(1) `w="12345"` `x="3"` (both are string values which can convert to numeric)

```
z=w+x      // z="123453" (string concatenation)
z=0+w+x    // z=12348 (numeric - see Forcing A Presumption, above)
z=w-x      // z="1245" (string reduction)
z=0+w-x    // z=12342 (numeric - see Forcing A Presumption, above)
z=x*w or z=x/w // error; string values cannot be multiplied or divided
z=1*x*w    // z=37035 (numeric) (numeric - see Forcing A Presumption, above)
z=1*x/w    // z=4115 (numeric) (numeric - see Forcing A Presumption, above)
```

(2) `w="12345"` `x=3` (`w` is a string value which can be converted to numeric, `x` is numeric)

```
z=w+x      // z=12348 (numeric)
z=w-x      // z=12342 (numeric)
z=w*x      // z=37035 (numeric)
z=w/x      // z= 0.000243013365735115
```

- (3) `w="1" x="9" y=5` (w and x are string values which can be converted to numeric; y is a numeric value)

```

z=w+x+y    // z=24 (numeric)
z=x+w+y    // z=96 (numeric)
z=w+y+x    // z=15 (numeric)
z=y+w+x    // z=15 (numeric)
z=y+w-x    // z=14 (numeric)
z=x+w-y    // z=86 (numeric)
z=x-w+y    // z=14 (numeric)

```

- (4) `w="2" x="ABC" y=5` (w is a string value which can be converted to numeric; x is a string value which cannot; and y is a numeric value)

```

z=w+x+y    // z="2ABC5" (string)
z=y+w+x    // z="7ABC" (string)
z=w-y+x    // z="-3ABC" (string)
z=w*y+x    // z="10ABC" (string)
z=y/w+x    // z="2.5ABC" (string)

```

• It's Your Choice. I'll close this discussion of automatic data conversion by saying that it is not difficult to include error trapping routines which test the value of any variable to make sure that it is the type of variable (string or numeric) that you intend and not rely upon automatic data conversion at all. That's my preference – I guess that it makes me "feel" better. But, do know that you have a choice in the matter, and that you may be very content to use the methods described above.

- ! **ADDITION.** The plus sign, "+", adds numeric values. So, `var=1+3` results in var having a value of 4. Numeric values in the formula can be replaced by variables containing numeric values, so that `var=var1+var2` would result in var having a value of 49, if `var1=30` and `var2=19`. For this operator's string usage, see **concatenation**, above.
- ! **SUBTRACTION.** The minus sign, "-", subtracts one numeric value from another. So, `var=3-1` results in var having a value of 2. Numeric values in the formula can be replaced by variables containing numeric values, so that `var=var1-var2` would result in var having a value of 11, if `var1=30` and `var2=19`. But, see **Floating Cell Problem**, below. For this operator's string usage, see **reduction**, above.
- ! **MULTIPLICATION.** The asterisk, "*", multiplies one numeric value with another. So `var=2*4` results in var having a value of 8. Numeric values in the formula can be replaced by variables containing numeric values, so that `var=var1*var2` would result in var having a value of 570, if `var1=30` and `var2=19`.
- ! **DIVISION.** The leaning-to-right slash, "/", divides one numeric value by another. So `var=4/2` results in var having a value of 2. Numeric values in the formula can be replaced by variables containing numeric values, so that `var=var1/var2` would result in var having a value of 1.5789473..., if `var1=30` and `var2=19`.
- ! **THE FLOATING CELL PROBLEM.** Due to "floating cell" issues inherent in processor chips, SOMETIMES simple math routines, in my experience subtraction, particularly, using `num3=num1-num2` DOES NOT produce an exactly correct result. See this thread at WordPerfect Universe: <http://www.wpuniverse.com/vb/showthread.php?s=&threadid=1775>. This is true in Excel, Quattro Pro, and it is true in WordPerfect macros. Try this yourself in your spreadsheet program: 692 - 685.08, and be sure your column width is overly wide. I think you'll find the result to be 6.91999999999996, but, even if slightly different, I doubt that your result will be 6.92, the obviously correct result. So, something is amiss and the problem can manifest itself in WordPerfect macros when using simple numeric subtraction, `num3 = num1 - num2`. To test this anomaly yourself, copy the following code to a macro, name it as you will, e.g., `mathtest.wcm`, and then run the macro.

```

Application (WordPerfect; "WordPerfect"; Default!; "EN")
GetNumber(n1;"Enter a real number... try 692";"Testing Macro Subtraction")
GetNumber(n2;"Enter a 2nd number ... try 685.08";"Testing Macro Subtraction")
n3=n1-n2
MessageBox("Computation Result of n1-n2";n3)

```

Unless I am badly mistaken, n3 will equal 6.91999999999996 using the above macro code. The correct result should be exactly 6.92. As far as I'm able to tell, this problem relates only to subtraction, not addition, multiplication or division. So, when doing subtraction, it makes sense to massage the numbers to be sure that results are exactly correct. An example is shown in [Math.wcm](#), below.

J. Dan Broadhead offers this explanation:

This problem is not really Windows related, but is related to the computer processor. And it has to do with the fact that the computer internally uses base 2, and we (humans) use base 10. Most fractional base 10 numbers cannot be 100% precisely represented in base 2 as the exact same value. When the base 10 number is to be displayed, the internal base 2 number is converted back to base 10, and displayed, sometimes showing the imprecision problem. It is more apparent with certain values than others.

This is usually solved by rounding off the number after it is converted to base 10 for display. The problem lies in the "rounding off for display" operation.

This is not limited to just subtraction, but could potentially occur for any mathematical operations.

The internal numbers are not affected, they are still as precise as possible, but the display appears to be imprecise due to the rounding not working entirely correctly in all cases.

So the best advice is not to worry about it when performing the math operations, but to only worry about it when the values are going to be displayed someplace. Continual rounding after every math operation could eventually have an effect on the precision of the values.

! MASSAGING NUMBERS. This discussion assumes that "real" numeric values are used in the processes described herein ... or that [automatic data conversion](#) has occurred if possible, or that some other method has been used to insure and/or convert values to be numeric and not string.

! Integers (Whole Numbers). By definition, an [integer](#) (a whole number) has no "points" at all – it is 100, it is not 100.5 or 100.0. Results are not "rounded" to reach a whole number. If a non-numeric value is used with the command, macro operations will stop and an error message will be generated. If you want a math result to be an integer, use the Integer expression, like this (num1 and num2 can be numbers or variables containing numeric values):

num3 = Integer(num1+num2) Integer(50.6+50) num3=100	num3 = Integer(num1-num2) Integer(50.6-50) num3=0	num3 = Integer(num1/num2) Integer(50.6/50) num3=1
num3 = Integer(num1*num2) Integer(50.6*50) num3=2530	num2 = Integer(num1) Integer(50.6) num=50	

Note that 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.

! Working With Decimals. On the occasions that you need to do math which involve decimal values, it is essential that you have a working knowledge of [NumStr](#), [Strnum](#), [Substr](#), [StrLen](#), [Concatenation](#), and [Reduction](#). Click the links just shown for the Glossary (Chapter 10) descriptions, and also see [StrTransform](#), [StrTrim](#) and [StrToChars](#) in the Glossary for other useful tools. Links in the next paragraph are to links within this chapter. Briefly stated, these commands or procedures do this:

[NumStr](#) converts numeric values to strings and only works with numeric values. [StrNum](#) only works with string values and converts them to numeric. [Concatenation](#) joins two or more strings together; and [Reduction](#) reduces the content of a string by the content of another. These work with either numeric or string values: [SubStr](#) extracts part of the value; [StrPos](#) determines the position of particular character(s) in a value; [StrLen](#) determines the character length (including spaces) of a value. Each of these commands and techniques is valuable in working with decimal (including percentage) values.

Practical examples follow these preliminary notes. First, here's a bit more detail.

NumStr() converts a numeric value to a string value. If the value attempted to be converted is not numeric, error occurs and the macro will stop. NumStr has 2 realistic parameters but only 1 parameter is required – the numeric value to convert, which can be a raw numeric value or can be a variable containing one. If `var=0.005`, both the following produce the same result: `x=NumStr(0.005)` `x=NumStr(var)`. The optional parameter deals with decimal values ... those following a decimal, if any. The parameter can be 0 to 16. Rounding should occur for points beyond than the parameter value, breaking a 5 (up). So, `NumStr(var;0)` rounds the value to what amounts to a whole number string value, `NumStr(var;2)` rounds the value to a string value with 2 decimal points, if in fact there are any decimals to round. It doesn't "add" decimal strings (e.g., ".00") if none are present in the value being affected.

Windows 2000 and XP users should note: Apparently, changes made beginning with Windows 2000 sometimes cause NumStr to not round correctly, e.g., if `var=12345.555`, `x=NumStr(var;2)` should result in x having a value of 12345.56, but, instead, the result is 12345.55, even though the correct result obtains in earlier Windows versions. See the following link at WordPerfect Universe if you want to know more: www.wpuniverse.com/vb/showthread.php?s=&threadid=6961. So, you may want to consider using alternative means than NumStr in Windows2000 and higher. A correct result can always be obtained by breaking the whole and decimal portions of string values and then recombine them, as is shown in examples in this chapter and in **Chapter 9**, particularly in **Math.wcm**.

StrNum() is the opposite of NumStr() – it converts string values to numeric values. If the 1st string character is not a numeric value, error results and the macro will stop. The single parameter is the string to convert. The parameter can be a literal string, or it can be a variable containing a string. So, if `var="50.00"`, `var=StrNum("50.00")` produces the same result as `var=StrNum(var)`, the numeric value of 50.0. If a decimal is not present in the string, a decimal will not be presented in the converted numeric value. So, if `var="50"`, then the converted numeric value would have no decimals but would be the numeric value 50. StrNum can only work with numeric values – characters, including commas, are not numeric values. The only valid characters are +-1234567890 and "period" (decimal). If a non-numeric value is encountered in the string, it and everything following will be disregarded when using StrNum, rather like **Reduction**, above, by analogy. So, if `var="1234g"`, `x=StrNum(var)` will return a numeric value of 1234 to x.

StrPos() is used to determine the position of a particular character (or characters) in a string or numeric value. In working with decimals, the character will typically be a decimal, a "period". Two parameters are used: the string, which can be a variable, and the character to locate. The parameters are separated by a semicolon and the searched for character is between a pair of regular (not curly) quotation marks. If the character is not present, a numeric value of zero is returned, but, otherwise, the exact location of the 1st matching character in the string is returned. So, if `var="500.25"`, `x=StrPos(var;".")`, the value of x is 4, the 4th character. If `var="500"`, `x=StrPos(var;".")`, the value of x is 0 (zero) since the decimal is not present.

SubStr() is used to extract "substrings" from numeric or string values. It has 3 parameters: the value (commonly a variable); the numeric beginning point of the extraction within the value; and the numeric ending extraction point within the value. The beginning and ending points can both be variables, or can be formulas based upon variables, such as by using **StrLen()**, described below, and/or by using **StrPos()** described above. The variable "type" (numeric or string) is unaffected by the command and the return value type will be the same as the original value type.

Here are 2 examples, where `var="500.25"` (string value), or where `var=500.25` (numeric value):

```
x=SubStr(var;1;3) returns a value of "500" (or 500 if var is numeric) to variable x
x=SubStr(var;4;2) and x=SubStr(var;4;StrLen(var)) returns a value of "25" (or 25) to variable x
```

Let's throw in the StrPos() command, using the same var value ("500.25" or 500.25):

```
y=StrPos(var;".") x=SubStr(var;1;y-1) returns a value of "500" (or 500) to variable x
z=Substr(var;x+1;StrLen(var)) returns a value of "25" (or 25) to variable z
```

So, now we have the elements of the original value broken into 2 parts, neither of which include the decimal, making it possible to perform operations on these 2 components of the original variable,

whether numeric or text. The length of each element can be determined using StrLen(), further testing can occur as for the existence of leading zeros, or whatever. The "whatever" will be largely shown in the remainder of this chapter.

Please don't get the idea that above shows the only way to accomplish the same thing. I'd be surprised if there aren't better ways. Here's another method that accomplishes essentially the same thing, where var="500.25" or var=500.25:

```
y=strpos(var;".")
x=Integer(var)
z=substr(var;y+1; Strlen(var))
```

StrLen() determines the number of characters, including spaces, in string or numeric values. If var="0.05" (string) or var=0.05 (numeric), then x=StrLen(var) results in x having a numeric value of 4: the 1st zero, the decimal, and the 2nd zero and the 5. StrLen can be used in combination with other commands, such as in the example immediately above.

! **SPECIFIC TASKS.** Very often the examples which follow are taken from **Math.wcm** in **Chapter 9**. The illustrations may be more tangible if you have an idea of what the main dialog in Math.wcm looks like. In Chapter 8, those illustrations here: **Image showing control names**; and **clean images**.

! **Getting Rid of Commas.** Commas aren't numeric values. So, to work with string values which appear to be numeric values (i.e., they include commas), here's the way I originally did that in **Math.wcm**, when I wasn't aware of the **StrTransform** command. This routine assumes that var is a string value, which may contain commas, and that Label(vComma) is called from elsewhere in the macro:

```
Label(vComma) // this strips commas in strings which, if used with NumStr, produces
               erroneous values
x=strpos(var; ",")
If(x>0)
  While (x>0)
    y=substr(var; 1; x-1)+substr(var; x+1; strlen(var)) var=y x=strpos(var; ",")
  EndWhile
EndIf
Return
```

Explanation: x=strpos(var; ",") determines the location of the 1st comma in var, if a comma is present. If not, x will equal zero (0). If so, x will be greater than zero. The **While - EndWhile** routine strips away each comma using SubStr and rejoins the substrings until no commas are present. So, if the initial value of var is the string value "1,222,333,444.50", the resulting value will be "1222333444.50". With this routine, var=StrNum(var) results in var equaling the numeric value of 1222333444.5. Without this routine, var=StrNum(var) results in var having the numeric value of 1 (since commas and what follows them are disregarded when using StrNum, **described above**).

As you can see, the above process relies upon **concatenation**. But, a simpler (but not the simplest – see below) is to use **reduction**, as follows:

```
Label(vComma) // this strips commas in strings which, if used with NumStr, produces
               erroneous values
x=strpos(var; ",")
If(x>0)
  Repeat
    x=strpos(var; ",")
    var=var-" ,"
  Until (x=0)
EndIf
Return
```

Explanation: similar to the concatenation example, except that commas are removed from variable var using reduction until no commas are left.

But, *the simplest* way to accomplish the same thing is to use **StrTransform**, as is done in the revised **Math.wcm**, like this:

```
var=StrTransform(var;";";"")
```

All commas are replaced by "nothing" in one simple phrase of code! Why didn't I use the above in the first (or second) place? I wasn't aware of the StrTransform command until recently!

! Converting a Numeric Value to a String to "Look" Right in a Document.

! Adding Text, Generally. If, after working with numeric values in a macro, numeric values will need to be stuck into a "real" document you are working on (or, into a dialog the user will see), then it may be desirable to modify the numeric values so that they "look right". If working with regular numbers, you will likely want commas inserted where they belong, and, if you want decimal values shown, you may only want to show 2 decimal values. If you want money values, you will want some "money" sign, e.g., a \$ (dollar), £ (pound), or i (Euro) sign, or if you want a percentage value displayed, a % (percent), sign added to the value before it is written into a document.

To add a character, simply do so: `xvar="$"+var` or `xvar=var+"%"`

Note that I've not changed the value of variable "var" in either of the above, just in case I need to use such values in later computations. Instead, I've used "xvar" as the variable which can be used for display purposes in a dialog or when writing the variable to text. Of course, do what makes sense in the context of your macro.

! Adding commas. Chances are that you won't want your document to display a numeric value such as 1234565000. Instead, you'd want it to be 1,234,565,000. While other ways are possible, here's one that works for me – **Label(NewNum)** is extracted here from **Math.wcm**, below. The macro code below assumes that variable "var" contains a numeric value, and that **LabelNewNum)** is called from elsewhere in the macro:

```
Label(NewNum) // this routine massages numeric values into new string values
If(var=0)
Return
EndIf
x=StrPos(var; "-")
If(x=1)
var=Substr(var; 2; strlen(var)) NegNum=1
EndIf
x=StrPos(var; ".")
Switch(x)
Caseof 0: tNum1=var Call(mainNum) tNum2=""
Default: tNum1=Substr(var; 1; x-1) Call(mainNum) tnum2=substr(var; x; strlen(var))
x=strlen(tnum2)-1
If(x<varD and vFinal=1)
Repeat
tnum2=tnum2+"0" x=strlen(tnum2)-1
Until(x=varD)
EndIf
EndSwitch
var=tNum1+tNum2
If(NegNum=1)
var="-"+var
EndIf
Discard(NegNum; tNum1; tNum2)
Return
Label(mainNum) // this routine creates commas in the main number from Label(NewNum)
If(StrLen(tNum1)<=3)
Return
EndIf
z=Integer(StrLen(tNum1)/3)
```



```

newvar=""
Repeat
  y=SubStr(tNum1;StrLen(tNum1)-2;3)
  tNum1=SubStr(tNum1;1;StrLen(tNum1)-3)
  If(StrLen(tNum1)>3)
    newvar=","+y+newvar Else newvar=tNum1+","+y+newvar
  EndIf
  If(StrLen(tNum1)>3)
    z=Integer(StrLen(tNum1)/3) Else z=0
  EndIf
Until(z=0)
tNum1=newvar
Return

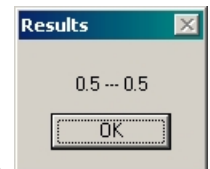
```

! Percentages. You've used your macro to produce a percentage, or, at least, which will become a percentage, by division. Your macro code might be:

```

Application (WordPerfect; "WordPerfect"; Default!; "EN")
var1=50 var2=100 var=var1/var2 var3=NumStr(var)
MessageBox("Results";var+" --- "+var3)

```



If multiplication, division, addition or subtraction produces a result less than 1, a leading "zero" and decimal will always be part of the numeric value returned, and so will the result of using NumStr(var) to convert the value to a string, both of which values (numeric "var" is at the left, string "var3" is at the right).

But, in your document, you want the 0.5 value to be 50% or 50.0% or 50.00%. Various means are available to accomplish that, but here's one way:

Massage the string value to determine its parts and then put them all back together again. Here's the previous code in **Label(MakePer)** in the **Math.wcm** macro below – the label is called from some other location in the macro and notice that it also calls Label(NewNum), shown above. Note that the variable being tested and worked over is a variable named "var" which contains a string value:

```

Label(MakePer) // this routine creates percentage string values; revised 6/4/2004 for scientific notation values
e=0
var=StrTransform(var;";";"")
var=strnum(var)
var=var*100 // a % value is always 100 times the non-percentage value
x=strpos(var;"e") // scientific notation routine
If(x>0)
  e=1
  y=substr(var;x+2;3) y=strnum(y) y=y+1
  var=substr(var;1;x-1)
  var=var+"."
  varq=strlen(var)
  If(varq<y)
    Repeat
      var=var+"0"
    Until (strlen(var)=y)
  EndIf
  x=strpos(var;".")
  If(x=0)
    var=var+"."
  EndIf
EndIf
If(e=0)
  var=numstr(var) // want to determine if that following the decimal is greater than 0, but not if scientific notation
EndIf

```

```

x=strpos(var;".") testn=substr(var;x+1;strlen(var)) testn=strnum(testn) // testn is rem w/o the "."
If(testn=0)
    var=substr(var;1;x-1) // If the remainder is 0, drops the ".0" from var
EndIf
If(e=0) // not if scientific notation
    var=strnum(var)
EndIf
Call(NewNum)
var=var+"%"
If(negNum=1)
    var="-"+var
EndIf
Return

```

! ERROR TRAPPING ROUTINES. Unless you build in routines to trap errors, when commands dealing with strings are used as though they were numbers, and vice versa, macro error may result causing the macro to stop or unintended results may occur. The 1st step is to insure that a user's intended "numbers" are numeric. An "O" (oh) is not a zero (0), as you know. A comma is not a numeric entry, either. So, the 1st thing to do is to test the user's data (such as from an edit box in a dialog) to insure the characters are "legal" if they are converted to a numeric value (as by using StrNum or in performing mathematical computations). The following are **Label(chkNum)** and **Function IsNumberString(InString)** contained in **Math.wcm**, below. It also tests for a "minus" sign in the 1st character position in the variable. It assumes that variable "var" contains a string value, and that Label(chkNum) is called from elsewhere in the macro:

```

Label(chkNum) // this routine determines if all characters are valid for numeric purposes; revised 6/4/2004
x=StrToChars(var;Keep!;".") x=StrLen(x) //this routine insures that only one decimal is in the string
If(x>1)
    q=1
    Return
EndIf
x=substr(var;1;1)
If(x="-")
    NegNum=1 var=substr(var;2;strlen(var)) Else NegNum=0
EndIf
Ret=IsNumberString(var)
If(Ret)
    Else q=1
EndIf
Return // If the variable contains non-allowed characters, q=1 says error is present

```

```

Function IsNumberString (InString)
RefString="1234567890., "
ForNext (Count;1;StrLen(InString);1)
    If (StrPos(RefString;SubStr(InString;Count;1))=0)
        Return (False)
    EndIf
EndFor
Return (True)
EndFunc

```

The 1st routine in Label(chkNum) makes sure that only one decimal is present in the string by using **StrToChars**. Since a decimal is "legal" in Function IsNumberString(InString), we need to be sure that not more than 1 decimal is present. So, variable "x" initially becomes the string length of all characters other than "." If the string length is 1 or 0, fine, but if more than 1, that's not a "legal" number (in string form at the moment).

The 2nd routine determines whether var is a "negative" number by extracting the 1st character of the string and assigning the 1st character's value to variable "x". If x = a minus sign, variable "NegNum" is

assigned a value of 1, and variable "var" is reassigned the portion of the string following the minus sign. Otherwise, variable "NegNum" is assigned a value of 0. Sometimes you may care, or not, whether a variable is "negative", and, if so, the "NegNum" value may be used following the point of the Call(chkNum) depending on whether you care if the value is negative.

Following that routine, the remainder of the code determines whether variable "var" contains any characters other than 1 2 3 4 5 6 7 8 9 0 or a period or a comma. This is done by calling the Function IsNumberString by use of the code, Ret=IsNumberString(var). If the variable contains any other characters other than those just mentioned, variable "Q" is assigned a value of 1. Note that, here, a comma has been treated as "legal" even though it is not, strictly speaking, for numeric purposes. A subsequent routine will eventually need to address the presence of commas. But, not now.

After this operation, macro flow returns to the point following the Call(chkNum). For example, in Label(getV), the call "shorthand" chkNum is used in place of Call(chkNum):

```
Label(getV) var=RegionGetWindowText("vMath.B2") If(var="") var="0" EndIf
chkNum If(exists(q)) RegionSetWindowText("vMath.S6";"2nd value, "+var+", is not legal")
RegionSetFocus ("vMath.B2") RegionSetEditSelection("vMath.B2") Return EndIf

If(NegNum=1) q=1 RegionSetWindowText("vMath.S6";"2nd value, "+var+", is not legal. Use
Subtract in regular date list instead.") Discard(NegNum) RegionSetFocus ("vMath.B2")
RegionSetEditSelection("vMath.B2") setResult Return EndIf

var=StrNum(var) If(vMethod="D") tempd2=var Else tempL2=var EndIf SecondN=var Return
```

Immediately following the chkNum call, if variable "Q" exists, a message is sent to the "vMath" dialog (particularly, to the "vMath.S6" control that the value entered in the dialog is not "legal"), and the macro does not continue beyond that point but instead the Return command ends the call. If variable "Q" does not exist, the macro proceeds.

Since, in the above example, since I DO care whether a negative entry was made in the dialog, variable "Q" is assigned a value of 1 and a message is again sent to the same "vMath.S6 " control, and, once again, the macro does not proceed past the point of the Return command which ends the call.

Otherwise, all tests being passed, the last line of the above code converts the string value to a numeric value. Some other stuff is done, too, not being relevant to this discussion.

Other error-trapping routines are certainly possible, but the above is one which I use again and again, and it works every time.

NOTES

NOTES

Chapter 7

Date Routines

Links: The **Chapter Name**, above, moves to the next chapter. 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.

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				
System Variables	Common Date Commands When Typing In Documents	Commands Not Covered	Massaging Dates	
Date Error Trapping Routines				

This chapter discusses some of the many date routines possible in WordPerfect macros. It is by no means complete as to all possibilities.

! **DATE SYSTEM VARIABLES.** Three or four **Date System Variables** are available, ?DateDay, ?DateMonth, and ?DateYear, and, for WordPerfect 8 and higher, ?DateWeekDay.

- **?DateDay** returns the numeric day of your computer's internal clock, 1-31. In WordPerfect 8 and higher, you can also use **?DateWeekDay** to return the string value of the actual day of the week, e.g., Wednesday.
- **?DateMonth** returns the numeric month of your computer's internal clock, i.e., 1 through 12.
- **?DateYear** returns the numeric year of your computer's internal clock, e.g., 2003.

So, should you wish to assign variable "vDate" the date that the macro runs in a mm/dd/yyyy date format, you could do this: vDate=?DateMonth+"/"+?DateDay+"/"+?DateYear and the value of vDate would be, for example, "7/2/2003 " (content varying based upon your computer's internal clock).

! **COMMONLY USED DATE COMMANDS WHEN TYPING INTO A DOCUMENT.** Although *many others* are available, those you'll probably use most often are DateCode, DateText and DateFormat(string):

- **DateCode.** This inserts the current computer clock date into a document, and keeps that date current if the document is opened on a later date. Although it is most used when simultaneously using the **Type** command, it is not part of the Type command but is entered as a stand-alone command.

So, a line of code might read,

```
Type("Signed on ") DateCode Type(" ")
```

- **DateText.** This inserts the current computer clock date into a document, and does not keep that date current if the document is opened on a later date. Although it is most used when simultaneously using the **Type** command, it is not part of the Type command but is entered as a stand-alone command.

So, a line of code might read,

```
Type("Signed on ") DateText Type(" ")
```

- **DateFormat(string).** This command identifies the particular date format you want to use in conjunction with either DateCode or DateText. Maybe you want the date format you intend to write to a document to be, "Now, on this ____ day of July, 2003,"

So, two lines of code might read,

```
Type("Signed on ") DateFormat("____ day of Month, Year(4)#") DateText [or]
```

```
Type("Signed on ") DateFormat("____ day of Month, Year(4)#") DateCode
```


If you want to use "ordinals" (e.g., 1st, 2nd, 3rd, 4th, etc.) for the "day" element and you want to have the "day" match your computer's system clock, you could do this:

```
x=?DateDay Call(setOrdinal)
DateFormat (vday+" day of , ")
DateText // or DateCode
Quit // or Return, if in a called label
Label(setOrdinal)
Switch(x)
  Caseof 1: vday=x+"st"
  Caseof 2: vday=x+"nd"
  Caseof 3: vday=x+"rd"
  Caseof 21: vday=x+"st"
  Caseof 22: vday=x+"nd"
  Caseof 23: vday=x+"rd"
  Caseof 31: vday=x+"st"
  Default: vday=x+"th"
EndSwitch Return
```

! DATE COMMANDS NOT COVERED. What's been said above covers date macro commands most "common users" will want to know. But, so that you'll be aware that many other **macro date commands** are available (and about which you can find information in the WordPerfect Macros on-line help file), know that these additional commands are available in WordPerfect 8.0 and higher:

DateAddDays (but see Math.wcm , below)	DateMonthName
DateAddMonths (but see Math.wcm , below)	DateOfMonthEnd
DateAddWeeks (but see Math.wcm , below)	DateOfNthDay
DateAddYears (but see Math.wcm , below)	DateOfNthWeek
DateAndTime (but see Math.wcm , below)	DateOfNthWeekday
DateDay	DatePart
DateDayOfYear	DateString
DateDaysInMonth	DateWeekday
DateDaysInYear	DateWeekdayName
DateIsLeapYear (but see below for Wp8 or later)	DateWeekOfYear
DateMonth	DateYear

J. Dan Broadhead informs us that only the following are available in WordPerfect 7.0: DateAndTime, DateDay, DateMonth, DateMonthName, DateString, DateWeekday, DateWeekdayName, and DateYear, and that none of these commands are present in WordPerfect 6.1 or 6.0.

! MASSAGING DATES. Several macro means are present to "massage" date values generated by various macro commands into string values. This shows only the means that I commonly use.

- **Converting Numeric Date Values To Text.** Recalling that one of my primary objectives is that my macros, where possible, work in WordPerfect 6.1 and higher, and noting that if WordPerfect 6.1 and/or 7.0 are avoided from consideration I might do this differently, this is the routine I commonly use to convert numeric date values to string date values. Note: *I avoid using Dialog Date Controls – they tend not to be cross-WordPerfect macro compatible, except in WordPerfect 8.0 and higher.* Instead, the dialogs I make use Edit Box Controls requesting date values in an mm/dd/yyyy date format, e.g., "3/3/2002", with the Edit Box maximum length being 10 (that covers all reasonably possible dates). With that background, where variable "var" contains the date string value of "mm/dd/yyyy", the following converts that value to an ordinary a date string value (e.g., July 2, 1943). Somewhere in the macro, Label(DateCnv) would be called to test a user's date entry in a dialog's Edit Box:

```
Label(DateCnv) // this tests and sets the numeric parts of a m/d/yyyy date and makes string
values
Call(dateChk) If(exists(q)) Return EndIf
// The above calls the error-trapping routine for a correct date format
x=strPos(var;"/") // this determines if an illegal pair of forward slashes have been used
```

```

If(x>0)
  q=1
  vMsg=var+" is not a legal date. Don't use consecutive '/' marks." setResult Return
EndIf
x=StrPos(var;"/") // If no forward slashes have been used, the date format is illegal
If(x=0)
  q=1 vMsg=var+" is not a legal date. Use a m/d/yyyy date format." setResult Return
EndIf
If(x>0) // This extracts the month portion of the date
  mo=substr(var;1;x-1) rem=substr(var;x+1;strlen(var))
  x=StrPos(rem;"/") // This extracts the day and year portion of the date
  day=substr(rem;1;x-1) year=substr(rem;x+1;strlen(rem))
  If(strlen(mo)>2 or strlen(day)>2 or strlen(year)<>4)
    // if month, day or year string lengths are wrong, this results in variable "Q" being declared
    // and equaling 1
    q=1 vMsg=var+" is not a legal date. Use a m/d/yyyy date format." Return
  EndIf
EndIf
Mo=Strnum(Mo) Day=Strnum(Day) Year=strnum(Year)
  // The values are all legal; this converts the strings to numeric values
If(year<1601)
  q=1 vMsg="Dates earlier than 1/1/1601 cannot be used. Change the date" Return
EndIf
  // The routines which are used below cannot be earlier than 1/1/1601; hence this error
  // routine
If(Mo=0 or Day=0 or Year=0)
  q=1
  vMsg="Don't use zeros for months, days or years. "+Mo+"/"+Day+"/"+Year+" is not a legal
  number." Return
EndIf
If(Mo>12)
  q=1
  vMsg=Mo+" is not a legal month. Use 1~12." Return
EndIf
Switch(Mo)
  Caseof 1: vMo="January"      Caseof 7: vMo="July"
  Caseof 2: vMo="February"    Caseof 8: vMo="August"
  Caseof 3: vMo="March"       Caseof 9: vMo="September"
  Caseof 4: vMo="April"       Caseof 10: vMo="October"
  Caseof 5: vMo="May"         Caseof 11: vMo="November"
  Caseof 6: vMo="June"        Caseof 12: vMo="December"
EndSwitch
vLeap=DatelsLeapYear(year) // this is used for determining what to do with February, below
Switch(vMo)
  Caseof "February":
    Switch(vLeap)
      Caseof False: If(day>28) q=1 EndIf
      Caseof True: If(day>29) q=1 EndIf
    EndSwitch
  Caseof "April": If(Day>30) q=1 EndIf
  Caseof "June": If(Day>30) q=1 EndIf
  Caseof "September": If(Day>30) q=1 EndIf
  Caseof "November": If(Day>30) q=1 EndIf
  Default: If(Day>31) q=1 EndIf
EndSwitch
If(Exists(q)) vMsg=vMo+" "+year+" does not contain "+day+" days." Return EndIf
  // above: vMsg will be used in another dialog which reports an erroneous date

```

```

var=vMo+" "+Day+", "+Year // no error resulting, var is reassigned to the resulting string value
Return
Label(dateChk) // this tests for m/d/yyyy string legal date characters
ret=IsNumberDate(var)
If(Ret) Else vMsg=var+" is not a legal date. " q=1 EndIf Return
Function IsNumberDate (InString)
  RefString="1234567890/"
  ForNext (Count; 1; StrLen(InString); 1)
    If (StrPos(RefString; SubStr(InString; Count; 1))=0)
      Return (False)
    EndIf
  EndFor Return (True)
EndFunc
// This ends the date error trapping routine – if the date is erroneous, variable Q will be declared
// and will have a value of 1; otherwise, no error (Q will be undeclared) will result and the
// elements of the date string will be used to create the string date

```

Following the called Label(dateCnv), and depending on the results obtained thereby, macro flow is returned to the point of the call.

For example, in Math.wcm, **one such label** is this:

```

Label(doDates) // this routine is called for "Dates", not "Numbers" or "Lawyer dates"
var=RegionGetWindowText("vMath.B1") tempD1=var
var2=RegionGetWindowText("vMath.B2") tempD2=var2
If(var="")
  q=1
  RegionSetWindowText("vMath.S6"; "Nothing to compute - enter date in top box")
  RegionSetEditSelection("vMath.B1") setResult Return
EndIf
x=substr(dsel; 1; 1)
Switch(x)
  Caseof "D"; "W"; "Y"; "M":
    If(tempD2="")
      RegionSetWindowText("vMath.S6"; "Nothing to compute - enter date in second box")
      RegionSetEditSelection("vMath.B2") setResult Return
    EndIf
  Default:
EndSwitch
Call(dateCnv)
If(Exists(q))
  RegionSetWindowText("vMath.S6"; vMsg)
  RegionSetFocus("vMath.B2")
  RegionSetEditSelection("vMath.B2") setResult Return
EndIf
FirstN=var tempd1=mo+"/"+day+"/"+year // etc. ... the call continues since "tests" were passed

```

NOTES

NOTES

Chapter 8

DialogShow & Callbacks

Links: The **Chapter Name**, above, moves to the next chapter. 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.

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				
Changing Perceptions	DialogShow Generally	DialogShow Syntax	DialogShow Without A Callback	
DialogDismiss & DialogDestroy	Why Callbacks?	Commands Not Covered	Callback Basics	
General Definition & Elements	One Callback At A Time	Region Names, Generally	RegionGet... Commands	
RegionGetCheck	RegionGetSelectedText	RegionGetWindowText	Putting It All Together	
Sample RegionGet... Code	Other Region... Commands	RegionShowWindow	RegionResetList	
RegionAddListItem	RegionRemoveListItem	RegionSelectListItem	RegionSetCheck	
RegionSetFocus	RegionSetWindowText	Making A Double-Click End A Callback	WordPerfect Version Control	
Commands At The Beginning of	Learning By Example	Example 1 Simple Beginning	Example 2 Building on Example 1	
Example 3 From Math.Wcm	Using DialogShow During A Callback	Example 4 WaitMessage		

! **TROUBLE IN RIVER COMMON PERSON'S CITY.** In my initial draft of this manual, I wrote this chapter (and corresponding sections in Chapter 10, Glossary, as well as other relevant chapters and sections), totally based upon my experience and what I'd picked up here and there but without having learned from any teacher, and certainly in no systematic way. As I said in the Preface, I've learned what I've learned on my own and on an "as-needed" basis. Over several years' time, I'd learned to write DialogShow and corresponding callbacks, as well as other commands which will be discussed in this chapter, which worked the first time, every time, and always for me without fail. I was happy with myself and what I had learned to do without a teacher. I was boldly ready to present my then existent macros manual based upon that experience.

Still, as to the initial draft of this manual, I was well aware that I am a "Common User" without any knowledge other than gained from experience and picking up this and that from here and with the single practical object, "does it work?" That being so, I invited up to five interested persons to review the initial draft of this manual at WordPerfect Universe on December 27, 2002: <http://www.wpuniverse.com/vb/showthread.php?s=&threadid=7968>. There were three takers – see the footnote on the first page. In the original post, I had the foolishness to say, "In a week or two, I'll be posting the finished document at my website for free download/viewing, as may be desired by anyone who has an interest in doing so." Ha! Now, more than a year later, here it is.

Far and away, the most complete and comprehensive review and instruction presented was very generously, uncommonly, and, to me, astonishingly submitted by J. Dan Broadhead. This paper is satiated with changes, suggestions, and corrections based upon his input. This chapter, particularly, has undergone more radical change than any other since, based upon his teaching, I learned that much if not most of what I'd been doing was being done incorrectly, even though it "worked for me".

How to deal with this "new" knowledge presented a formidable challenge – should I rewrite the chapter from scratch (as well as related items in Chapter 9, Glossary), even though what I would be saying wouldn't be coming from me (based on what I know or think that I know), or should I just leave well enough alone? The former option would go against the grain of what I said, and continue to say, in the Preface: "So, get out your 'book' and write this down: You don't need to be a macro guru ... if a macro works, it works." However, the latter option would present information which I now know is contrary to orthodox teaching from **a person who knows PerfectScript** perhaps better than anyone else on the planet. He should – he wrote much of it and was the last person at Corel who did.

After a fierce internal debate <grin>, I decided that it would be improper to present this chapter as it was originally written, but that it would likewise be improper merely to parrot J. Dan's voluminous comments – I'm talking scores of pages of comments. So <groan>, I decided that, once again it was time to try to relearn stuff again and then present it as correctly as I was able, but still based on what I had experienced that actually "worked." Did you ever have to face up to the daunting challenge of relearning what you knew in Wp5.1 DOS macros to for WordPerfect Windows macros? I mean, at least Wp5.1 DOS had excellent documentation – do you remember how good those old Wp5.1 DOS manuals were?! So, that's what I've tried to do – relearn – and this revised chapter is hopefully much closer to the mark than what I'd said in the earlier draft! At the same time, I'll also mention some of my prior bad practices in case you may have picked up similar habits along the way.

! **DIALOGSHOW(), GENERALLY.** Most of this chapter focuses upon using DialogShow with callback routines. However, before getting into that, to avoid confusion, it's probably best to discuss the use of DialogShow, generally, and, particularly, the use of DialogShow without a callback.

- **Syntax.** As is particularly described in Chapter 10, Glossary, DialogShow is the command that causes a dialog to be displayed during the running of a macro. In WordPerfect 8.0 and higher, the syntax is:

DialogShow("DialogName";"WordPerfect"; [callbackLabel]; ["B1"]). Strictly speaking, only the first parameter is required (the string name of the dialog), but, as practical matter, you'd also want to use the second parameter, "WordPerfect", this being the "parent" program in which the dialog is being used. I don't recall seeing any WordPerfect macro written by others which did not include that parameter, so, just do it! The third optional parameter identifies the name (no quotation marks) of the callback label, if one is used at all. According to WordPerfect on-line help, the last optional parameter can be used to identify the control within a dialog which is to receive initial focus, although I have no experience with that parameter at all. I understand that the initial focus parameter was added in WordPerfect 7.0.

Normal syntax with no callback: DialogShow("DialogName";"WordPerfect")

Normal syntax with a callback: DialogShow("DialogName";"WordPerfect";cbDialog) (where cbDialog is the name of the callback label, e.g., Label(cbDialog).

- **Use of DialogShow Without A Callback.** According to WordPerfect's on-line help, and, better, J. Dan Broadhead, it's pretty simple: (1) DialogShow("vDialog";"WordPerfect") makes the dialog visible; (2) Clicking on a Push Button control (or a Windows "Close" [x] button) closes the dialog; (3) Doing so automatically dismisses the dialog and a value is automatically assigned to an internal and automatic variable, MacroDialogResult – the value assigned to MacroDialogResult is the value of the button that closes the dialog: "CancelBttn" and Windows "Close" buttons assign a numeric value of 2; an "OKBttn" assigns a numeric value of 1; "Other" Push Buttons assign the string name of the Push Button control, e.g., "OtherBttn"; (4) IF the value of MacroDialogResult is 1 or the string value of any "Other" Push Button control, AND IF any of the dialog's controls, e.g., a List Box, Combo Box, have been associated with variables in their respective control properties, any such variables will be declared (or updated) and the value in the control will be assigned to the associated variable(s); but (5) IF the value of MacroDialogResult is 2, no such variables will be declared (or updated).

Consequently, to the extent that a dialog's controls have variables associated with them in their respective control's properties, if the automatically assigned value of MacroDialogResult is anything other than 2 (numeric), those variables will either be declared or updated and the value assigned to each will be value of the particular control.

So, the point to remember here is that you don't need to use a callback dialog merely to obtain values of variables IF the variables are associated with corresponding controls in the dialog. And, if all that is intended when using DialogShow() is to display a help or other information dialog (e.g., a dialog showing that either user or macro error has occurred in some respect), and the obtaining of variables within a dialog is not involved at all, DialogShow() without a callback is all that is needed in such circumstances.

- Use of DialogDismiss() and DialogDestroy(). In WordPerfect 6.1, DialogShow() could only be used with Dialog Editor Dialogs, but, in WordPerfect 7.0 and higher, DialogShow() could be used in both Dialog Editor and Dialog Define dialogs. For WordPerfect 7.0 and higher, orthodoxy is as follows:
 - DialogDestroy() is used to remove an "old style" dialog (i.e., one written from the keyboard using the DialogDefine() command [aka "text" or "DialogDefine dialog"]) and/or a "new style" dialog (i.e., one created using the Dialog Editor [aka "Dialog Editor dialog"]) from memory.
 - DialogDismiss() is used to close a dialog opened with a DialogShow() containing a callback.
 - Ordinarily, don't use DialogDestroy() following a DialogShow() command NOT containing a callback.

Notwithstanding orthodoxy, over the years I'd formed intentional habits inconsistent with the above. My memory is not completely clear, but I recall that negative consequences sometimes occurred when dialogs in some WordPerfect versions were not "destroyed" by DialogDestroy, particularly in macros containing many Run() commands. To deal with that, I'd formed habits like that described below.

In using DialogShow() with callbacks, I would habitually use code like the following (where variable v had been assigned the value of ?MajorVersion):

```
vLoop=False vQuit=False DialogShow("v50a";"WordPerfect";Cb50a)
Repeat Until (vLoop) DialogDismiss("v50a";1) If(v>6) DialogDestroy("v50a") EndIf
```

Quite possibly (and I'm sorry that I'm unable to be specific but I have no present means of verifying what actually did happen), I may have used dialogs containing the same name in variously "run" macros. Even though one might reasonably think the following statement in WordPerfect's on-line help means *exactly* what it says, "A dialog created with the Dialog Editor, without a callback, is *removed from memory* when any button is pushed. Subsequent calls to the dialog create an error (the macro system no longer knows anything about the dialog)," I *now* understand that one should be careful about taking WordPerfect's on-line macro help at face value! In fact, a dialog created with the Dialog Editor *does* remain in memory, and will only be *removed* from memory by a DialogDestroy() command (at least, in some versions of WordPerfect – I apologize for not being able to be specific as to versions). At least in WordPerfect 7.0 and higher, if a parent macro contains a Dialog Editor dialog named, "vTips", and a child macro run from the parent macro also contains a dialog with the same name, if the parent macro's dialog has not been "destroyed", then error can result.

About this "issue" (I'd call it a "problem"), J. Dan Broadhead observes:

If a dialog is not destroyed (by calling DialogDestroy for Dialog Editor callback dialogs, or for all DialogDefine dialogs), then they stay in memory, taking up resources. When you begin to use the "Run" command, each subsequent macro shares the operating environment of the previous macro, and any dialogs not destroyed continue to reside in memory, and calls to DialogDefine() will cause an error because the dialog still exists in memory. Also, if this second macro had a dialog defined in its header (by the Dialog Editor) with the same name as the dialog still in memory from the previous macro, then calls to DialogShow() could also potentially cause an error because of the ambiguity over which dialog you mean to show.

In using DialogShow() without a callback in WordPerfect 6.1, to be sure that a dialog was removed from memory, I would systematically use code like this...

```
DialogShow("Help1";"WordPerfect") DialogDismiss("Help1";1)
```

... where an "OKBttn" was the only Push Button and the dialog was essentially a glorified Message Box.

Very clearly, such habits of mine were and are inconsistent with orthodoxy! As will be seen below, I've wholly accepted J. Dan Broadhead's teaching as to using DialogDismiss within the callback for DialogShow() with callbacks. But, if Orthodoxy doesn't work for some reason in WordPerfect 7.0 or later try using DialogDestroy(), or in WordPerfect 6.1 try using DialogDismiss("vDialog";1) (if the dialog contains an "OKBtn" Push Button control), and see what happens. The main thing: Figure out what works for you and DO that which does.

- DialogShow/DialogDismiss/DialogDestroy – Behind The Scenes. Before proceeding, and at the risk of this "Common Person's" manual becoming overly technical, the following comments by J. Dan Broadhead were very instructive to me as to the "why's" and "what's" of what's happening "behind the scenes" with DialogShow() commands, as well as understanding the implications associated with using DialogDismiss() and DialogDestroy(). Some of his remarks refer to content present in a draft version of this manual, no longer stated here, but you'll get the picture without those earlier draft contents. I'm thinking that his discussion would benefit you as much as it did me, and that you'd like to have it. So, take three deep breaths ... and dive in!

DialogDestroy/DialogDismiss/DialogShow

Regardless of version, only use DialogDismiss when using a *callback dialog*. Using it for a non-callback dialog could potentially change the output from the dialog.

Here is how this works.

Suppose you have a dialog, which you created using the dialog editor, called "dlg".

The dialog editor creates a "binary" (non-readable) form of the dialog, which is stored in the header of the macro file.

Lets first examine a non-callback dialog.

- 1) In your macro, you would first do a DialogShow() command. This command searches memory for a loaded dialog definition. If one is not found (there won't be one found in the case of dialog editor dialogs [not always true, but true enough for now]), then the macro header is searched for the dialog definition. When found, the dialog definition is loaded into memory.
- 2) Then, the dialog definition is used to create (internal to Windows) the dialog, but it is not yet visible.
- 3) Then for each control on the dialog, the variable associated with each control is located, and its value (if any) is placed into the control. If the variable does not exist or does not have a value, then the control is left alone.
- 4) The dialog is now made visible, and the macro stops to wait for the user to click one of the buttons on the dialog.
- 5) When the user finally clicks on a button, then the macro system internally calls DialogDismiss(), and passes the name of the button that the user clicked on to this command as the second parameter. So this is the name of the control used to close (dismiss) the dialog.
- 6) The DialogDismiss command first hides the dialog, and then looks at the control passed to it. If that control does *NOT* have the Cancel! style on it, then the values of all the controls on the dialog are each extracted, and placed into the variables associated with each control.
- 7) If the control *did* have the Cancel! style on it, then the variables are not updated.
- 8) The DialogDismiss command then places the name of the control that was used to close/dismiss the dialog into the special MacroDialogResult variable, and then goes back to the DialogShow command that your macro called.
- 9) The DialogShow command returns to your macro ready for the next command after the DialogShow command.
- 10) At this point, your macro could call DialogDestroy if you want, because the dialog definition is still in memory.

11) Your macro looks at the special MacroDialogResult variable to see how the dialog was closed/dismissed by the user. If you did call DialogDestroy, then the value in the special MacroDialogResult variable placed there by the internal call to DialogDismiss has been replaced by the result from calling DialogDestroy, and your macro cannot tell how the dialog was closed.

12) If you call DialogDismiss again, the dialog will already be gone (its been closed). I'm not entirely certain of exactly what will happen, but because the dialog, then the macro system cannot get the values from the controls on the dialog, and cannot update the associated variables (there are no controls, and it cannot tell what variables used to be associated with the dialog). MacroDialogResult may or may not be set with the name of the control that you pass as the second parameter.

But since DialogDismiss was already called for you by DialogShow, then there is no need to call it again, and its results could potentially be unpredictable.

Now suppose that this is a callback dialog. I'll add or remove steps to the above sequence. Not all of the callbacks mentioned are sent in all versions.

2a) After the dialog is created internal to windows, your callback routine is called to tell you that it was initially created.

3) Same as above. The controls are loaded with their initial values.

3a) Before the dialog is made visible, your callback routine is called again, to tell you that all the controls have been initialized.

4) The dialog is made visible, but the DialogShow command does not wait for the user.

4a) Before the next command in your macro after the DialogShow command is called, your callback is called several times again, telling you macro that the dialog has been sized and positioned on the screen, and that it has received focus. This are only send starting in version 8.

4b) The next command after DialogShow in your macro gets called. At this point, it is your responsibility to loop or something else so that your macro does not end, but somehow waits for the user to close the dialog (or at least attempt to close the dialog).

4c) In response to the user clicking a button on the dialog, or otherwise playing with the fields on the dialog, your callback routine is called to tell your macro what is happening.

It is the responsibility of your callback routine to watch for the buttons on the dialog to let the user close the dialog.

If the user clicks on a button, the dialog is **NOT** hidden. It remains visible, and your callback is called.

You must respond in the callback to this button click.

5) If you want to close the dialog, then **IN YOUR CALLBACK** you should call DialogDismiss, and pass as the second parameter, the name of the button that the user clicked on. Do it now, in your callback, and don't do it later in the code after the DialogShow.

6) Same as above. The DialogDismiss command hides the dialog, and then either updates the variables from the controls on the dialog,

7) or doesn't depending on the control name passed to it.

8) Same as above. The control name passed to DialogDismiss is placed into the special MacroDialogResult variable, but this time it goes back to your callback where you called it.

8a) Your callback must now set a variable or otherwise signal your main code, that it is time to exit from the looping that it is doing. You now return from your callback.

9) Your looping code in the main macro somehow sees that it should exit, and it drops out of the loop and continues with the next commands after the loop.

10) Same as above.

11) Same as above. Your macro looks at the special MacroDialogResult variable to see how the dialog was closed/dismissed by the callback (not the user). If you did call DialogDestroy, then the value in the special MacroDialogResult variable placed there by the call to DialogDismiss in your callback has been replaced by the result from calling DialogDestroy, and your macro cannot tell how the dialog was closed.

12) Same as above.

Go it so far? Now how does a dialog created by DialogDefine differ, both for non-callback and callback dialogs?

I'll add or remove steps to the above sequences where appropriate.

0) Your macro calls DialogDefine and DialogAdd... commands to create a dialog definition that is stored in memory.

1) Same as above. Your macro calls DialogShow. This time, the definition of the dialog is already in memory, and the macro system does not have to look in the macro file header.

2-7) Same as above for either non-callback or callback dialogs.

8) Same as above. MacroDialogResult is updated. Then the DialogDismiss command either returns to the DialogShow command (for a non-callback dialog) or returns to your callback (for a callback dialog).

9) Same as above for either non-callback or callback dialogs.

10) Since the dialog still exists and the dialog definition still resides in memory, then you could call DialogDestroy if you want to, or not, depending on what you might want to do with the dialog later on. If you choose to call DialogDestroy, then see note in #11 below.

11) Same as above for either non-callback or callback dialogs.

If you did call DialogDestroy, then the value placed into the special MacroDialogResult variable by the call to DialogDismiss has been replaced by the result from calling DialogDestroy, and your macro cannot tell how the dialog was closed.

If you choose to call DialogDestroy in #10, then you should do something first. First, you should copy the current contents of MacroDialogResult to a temporary variable in order to preserve its value. Then call DialogDestroy. Then assign the temporary variable back to the special MacroDialogResult variable, or else just examine the temporary variable instead of MacroDialogResult, in order to see how the dialog was closed/dismissed.

12) By now, the dialog and its definition in memory may or may not exist.

As in the dialog editor case, I'm not entirely certain of exactly what will happen if you call DialogDismiss again. Because the dialog and its definition is gone, then the macro system cannot get the values from the controls on the dialog, and cannot update the associated variables (there are no controls, and it cannot tell what variables used to be associated with the dialog). MacroDialogResult may or may not be set with the name of the control that you pass as the second parameter.

And since DialogDismiss was already called for you by DialogShow for non-callback dialogs, then there is no need to call it again, and its results could potentially be unpredictable.

However, if the dialog definition does still exist because you did **NOT** call DialogDestroy in step #10, then the macro system would know which variables to update. And since the dialog with its controls still exists, then the associated variables would be updated again, and MacroDialogResult will again be set with the name of the control that you pass as the second parameter.

13) If you want to display the dialog again in the macro, then if you called DialogDestroy in step #10 above, you will need to call step #0 again to recreate the dialog definition.

If you did not call DialogDestroy in step #10, then you can just start with step #1 the next time, because the dialog definition is still in memory.

If you can follow all that, then you are doing great. Now I need to throw one more common mistake into the explanation.

A common mistake for callback dialogs, is to bypass DialogDismiss, and just call DialogDestroy. If you have been able at all to follow the above explanations, then you might be able to see the implications of doing this.

The primary purpose for DialogDismiss, is to cause the variables associated with the controls to be updated when the dialog is closed. Somewhat as a side effect, the dialog is physically hidden.

If DialogDismiss is not called, then the dialog remains visible, it does not get destroyed, its definition is not deleted from memory, the associated controls are not updated, and MacroDialogResult is not updated to indicate how the dialog was closed (because it has not been closed yet).

In an attempt to compensate for this, then another common mistake is sometimes committed, which is to call DialogDestroy instead. DialogDestroy will cause the dialog to be hidden and then destroyed, and the dialog definition is removed from memory.

****BUT****, the associated variables were never updated, so all values in the controls on the dialog are lost, and MacroDialogResult is never updated to indicate which control was used to close the dialog.

Ah, one more point. When a dialog is created and the dialog has the OK! and Cancel! styles, then OK and Cancel buttons are automatically created for you and placed onto the dialog without you having to name and create them yourself. In this case, the OK button has a name of 1, and the Cancel button has a name of 2. These values can then be used in the DialogDismiss command as the second parameter to indicate which button was used to close the dialog. If you create your own OK and Cancel buttons, then you must name them when you create them, and you should use their names in the DialogDismiss command instead of using 1 and 2.

And you could potentially have several Cancel or OK buttons if you want. And always remember, that if you pass the name of any button that has the Cancel! style on it, then the associated variables will ***NOT*** be updated. In all other cases (whether the button has the OK! style on it or not), the associated variables will always be updated.

So the moral of the story is:

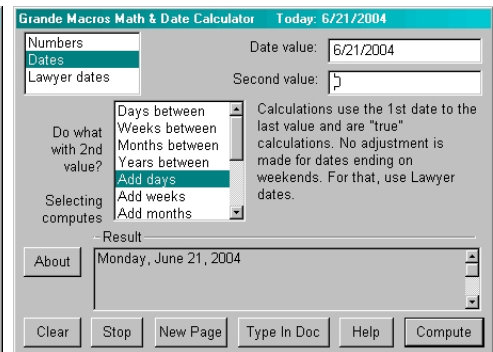
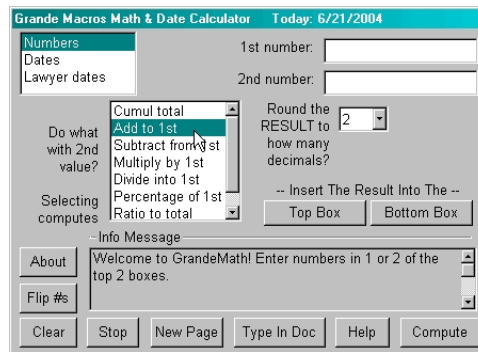
- a) Always call DialogDismiss for callback dialogs, preferably in the callback, and don't call DialogDismiss for non-callback dialogs.
- b) There is no particular need to call DialogDestroy. You can call it if you want. But if you do call it for DialogDefine dialogs, but you will need to call DialogDefine again before you can display it again.

Gulp! Now you can come up for air! My advice is to read, and re-read, the above until it makes sense to you, as it eventually will!

! WHY CALLBACKS? Callback routines provide the means of creating WordPerfect macro dialogs which are flexible, dynamic and non-static. A **DialogShow** command without a Callback Routine presents a "static" (non-changing) dialog, but, a DialogShow command WITH a callback routine allows the dialog to be fluid, immediately and dynamically change, and still remain visible, based on how you set it up.

"Seeing" this may be more meaningful to you than my verbal description. Here are two pictures of the same main dialog used in Math.wcm, Chapter 9. Even though the dialog is the same, it's appearance is quite different:

I'll not discuss the differences here, but, as you can see, as the selection in the upper left List Box control changed from "Numbers" to "Dates", the contents (and visibility) of other controls in the dialog changed, as well. These images are referred to in Chapter 6, [Specific Tasks](#).



Such types of things cannot occur using a `DialogShow()` command without a callback parameter.

As I said previously, if all you intend that a dialog do is obtain user input from controls and no need exists for the dialog to change during that process, you don't need to use callback routines at all, assuming that you've associated the various controls with variables in each control's properties. Either non-callback or callback dialogs will accomplish that if the controls in a dialog are associated with variables. In the former, variables will be declared or updated and assigned values when a non-cancelling Push Button is clicked, e.g., an "OKBtn".

But, if and when you want to do something "fancy", you'd want to use `DialogShow()` with a callback routine. The remainder of this chapter discusses how you can make such things happen.

! **NON-COVERED CALLBACK COMMANDS.** Other than brief discussion in Chapter 10 for [CallbackWait](#) and [CallbackResume](#), those commands are not developed in this paper. Since those commands are not available in WordPerfect 6.1, and since, when possible, I want my macros to work in all WordPerfect versions from 6.1 through current, I don't use them. See your on-line Macros Help File for these commands. Below, I'll be describing the "traditional" use of callback routines in WordPerfect, 6.1 through current.

Some selected "Region... dothis/dothat" commands which provide such dynamic control are covered in this paper. But, numerous others are not:

RegionAddListItemByIndex, RegionEnableWindow, RegionGetChildCount, RegionGetClass, RegionGetHandle, RegionGetID, RegionGetListContents, RegionGetListCount, RegionGetListItemByIndex, RegionGetListSelectedCount, RegionGetModified, RegionGetName, RegionGetOrder, RegionGetPosition, RegionGetStyle, RegionGetType, RegionGetVariable, RegionIsEnabled, RegionIsVisible, RegionMoveWindow, RegionRemoveListItemByIndex, RegionSelectListItemByIndex, RegionSetBitmap, RegionSetEditSelection, RegionSetModified, RegionSetProgressPercent, RegionSetSelectedText, and RegionSetTabStops.

A "common user" will not likely need them – at least, this common user hasn't. See On-Line Macros Help.

! **CALLBACK BASICS.**

- **General Definition and Callback Elements.** Without getting more technical than either I have the expertise to convey or for which you have the need to know, a callback statement consists of a (1) a `DialogShow()` command containing a callback parameter, that being the name of the called label, e.g., `DialogShow("vTest"; "WordPerfect"; cbTest)`; (2) the callback label itself which contains commands which execute during the callback routine; (3) events you identify (such as the clicking of a particular Push Button control within the dialog) which end the callback routine – the dialog will "loop" continuously until such an event is encountered; and (4) commands which stop the loop, close the dialog, and end the call. A `DialogShow()` without a callback label parameter does not "loop" and the dialog is closed when any Push Button control is clicked. Clicking on Push Button controls in a callback routine does not end the loop unless you define that they do so within the callback label.

- **One Callback At A Time.** Subject to J. Dan Broadhead's notes, below, while a callback label is running, another `DialogShow` command using a callback cannot run at the same time. However, it is possible to use `DialogShow` during a callback to display a dialog which does NOT contain a callback. This is useful for error-trapping routines to identify user error during the callback and/or "tip" dialogs invoked by a user clicking a Button control. See examples in [Math.wcm](#), below.

At <http://jdan.com/perfectscript/new-wp9x.htm>, J. Dan Broadhead explains that what I said above is not true beginning with WordPerfect 9, Service Pack 4, though I've not yet attempted to apply what he mentions there:

In the past, when a macro was currently executing code in the macro due to a callback event, then no other callback event would cause its associated code to be executed until the current callback completes. This is no longer true. If a callback event occurs for a callback label different from any callback label that is currently executing, then that callback event will cause the current callback code to be suspended, and the new callback code will be called. This fixes the difficult problem of displaying a callback dialog from the callback code of a previous callback dialog. By restricting this to different callback labels than any currently active callback label, this guarantees that a callback will complete before another callback for that same callback label it is allowed to occur, which could interfere.

In reviewing this manual, he added:

This never got mentioned in the documentation. I think Corel had abandoned any attempt to update the documentation when they closed the Utah in the middle of version 9 development. Whatever documentation had already been changed before the Utah office closure was all that probably ever made it.

However, even using the older versions, you can do what you want and have another callback dialog displayed from a callback dialog. But it requires extra work. You have to make sure you don't wait for the second callback dialog in the callback of the first dialog. If you are interested in knowing how to do this, then let me know, and I'll go into more detail on this with you.

I've not accepted his kindly invitation (as yet), and the above is mentioned mainly to let you know that other options than what I said in the opening paragraph are possible for later WordPerfect versions, should you be interested in exploring them on your own. For purposes of this manual, I'm operating on the "one callback at a time" principle.

- **Region Names, Generally.** In callback routines, it will not be uncommon for you to need to use a "Region Name" to identify controls located within the dialog for the various Region... commands. Every control in a dialog has a name (e.g., "B1") and, unlike variables and label names, Control Names are case sensitive. A Region Name consists of (a) The case-sensitive dialog name; (b) a "period" which separates the Dialog Name from the Control Name; (c) the Control Name; and, (d), a pair of quotation marks which encloses everything just stated. So, for example, a Region Name would be "vTest.B1" where the dialog's name is "vTest" and the control's name in that dialog is "B1".

When you create a control using the Dialog Editor, it will always be given a default "Control Name". You can change that name to match your personal preferences. Also, see [Chapter 5's](#) discussion.

When using Region Names in Region... Commands, it's always the same. So, for the RegionGet... commands which immediately follow, or other Region... commands described later, the Region Name portion of the command is always "vDialog.B1", where "vDialog" is the dialog name and "B1" is a Control Name within the dialog.

- **Regionget... Commands.** This section considers the three "RegionGet..." commands you will likely use most often, [RegionGetCheck](#), [RegionGetSelectedText](#), and [RegionGetWindowText](#).

Many other "RegionGet..." commands are not covered in this manual – [RegionGetChildCount](#), [RegionGetClass](#), [RegionGetHandle](#), [RegionGetID](#), [RegionGetListContents](#), [RegionGetListCount](#), [RegionGetListItem](#), [RegionGetListItemByIndex](#), [RegionGetListSelectedCount](#), [RegionGetModified](#), [RegionGetName](#), [RegionGetOrder](#), [RegionGetPosition](#), [RegionGetStyle](#), [RegionGetType](#), and [RegionGetVariable](#).

I'm listing them here in case you want to check them out in the on-line Macro Help file or elsewhere. Some of these commands are not available in WordPerfect 6.1 or 7.0.

- **Difference Between WordPerfect 6.1 and higher WordPerfect versions.** Syntax changed in WordPerfect 7.0. So, vary your code as follows: In WordPerfect 6.1, use this syntax:

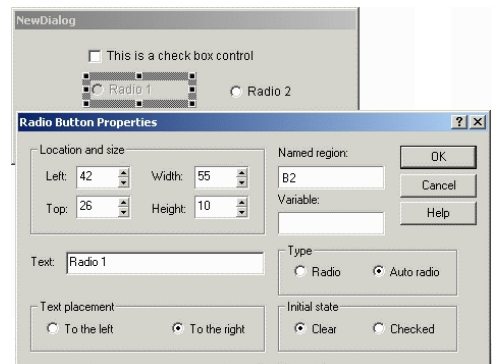
RegionGet...(var;"vDialog.B1") (for example). In WordPerfect 7.0 and higher, use this syntax: var=RegionGet... ("vDialog.B1") (for example). Note: var=RegionGet... will also work in WordPerfect 7.0 through current. Saying that, in WordPerfect 9.0, Service Pack 4, and through WordPerfect 12.0, if you use the WordPerfect 6.1 syntax, you will receive a compilation warning message similar to this one:

The command 'REGIONGETSELECTEDTEXT' is now a function. The procedure call format is obsolete, but should continue to operate 'as is' for now. To call it as a function, move this first parameter outside the parameter list, and assign the return value from 'REGIONGETSELECTEDTEXT'.

To avoid such messages in WordPerfect 9.0, SP4, and higher, use the "current" code described above. See **Errors and Warnings During Compilation** in Chapter 5 for more about that. The descriptions which follow use the preferred WordPerfect 7 and higher syntax. If you're using WordPerfect 6.1, change the syntax as described above.

- **RegionGetCheck.** Use **RegionGetCheck** to get the value of a Check Box Control or a Radio Button Control. The value is Checked! or Unchecked! and it is assigned to a variable thusly:

vB1=RegionGetCheck("NewDialog.B1") where dialog "NewDialog" contains a Check Box or Radio Button Control named region "B1". If "B1" is a Check Box Control and is checked, the value of "B1" is Checked!; if not checked, the value is Unchecked!. The same is true for Radio Button Controls. So, vB2=RegionGetCheck("NewDialog.B2") results in var having a value of Checked! or Unchecked!, depending on whether it is checked. Also, see **RegionSetCheck**, below.

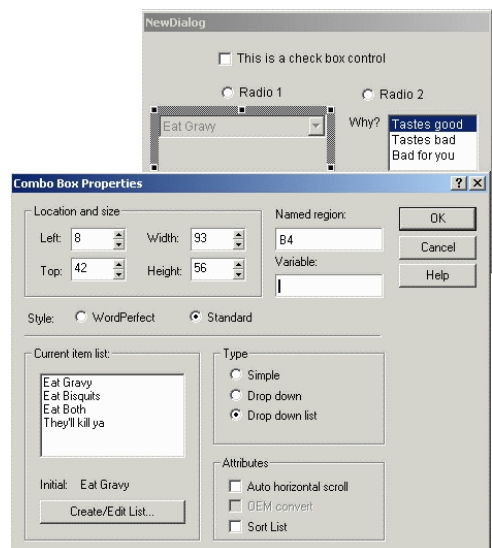


- **RegionGetSelectedText.** Use **RegionGetSelectedText** to get the value of List Box, Combo Box, or Counter Box controls that are included in a dialog. Counter Boxes are not covered in this manual, but see Macros On-Line Help.

Write your code like this:

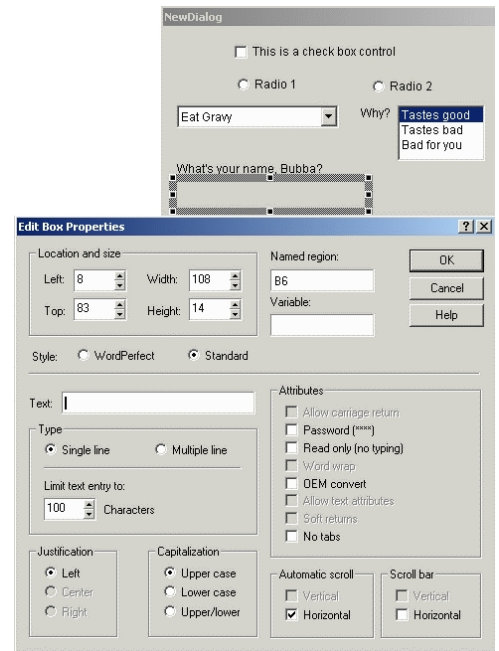
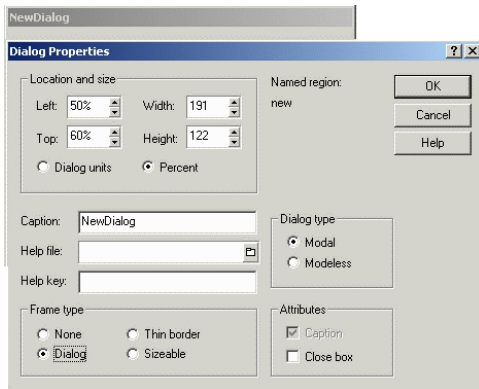
vB4=RegionGetSelectedText("NewDialog.B4") where dialog "NewDialog" contains a Combo Box Control named "B4" and/or vB5=RegionGetSelectedText("NewDialog.B5") where dialog "NewDialog" contains a List Box Control named "B5". Variable "var" will be assigned the item selected. See the images below:

Also, see **RegionResetList**, **RegionAddListItem** and **RegionSelectListItem**, below.



- **RegionGetWindowText.** Use **RegionGetWindowText** to get the value of Caption Bar text, a Static Text Box control, or an Edit Box control – i.e., text values in controls not having more than 1 value possible. Here's the exemplary code: vB6=RegionGetWindowText("NewDialog.B6") where dialog "NewDialog" contains an Edit Box Control named "B6".

To get the name of the Caption Bar, here's an example:
`vB0=RegionGetWindowText("NewDialog")` to get the text in the dialog's Caption Bar, shown below.



• Putting RegionGet... Commands All Together.

Below is a simple macro which illustrates all that's been said above. The dialog's name is "NewDialog" and you can name the macro anything you want. Although the dialog was originally made using the Dialog Editor, here it's been copied and pasted into this document so that you can copy it and run it for testing yourself. When you run the macro, you will get a message box like that shown below (depending on your selections and Edit Box entry):

`vB0` is the value of the dialog's Caption Bar;

`vB1` is the value of the dialog's Check Box control;

`vB2` and `vB3` are the values of the Radio Button controls;

`vB4` is the value of the Combo Box Control; `vB5` is the value of the List Box control; and `vB6` is the value of the Edit Box control.

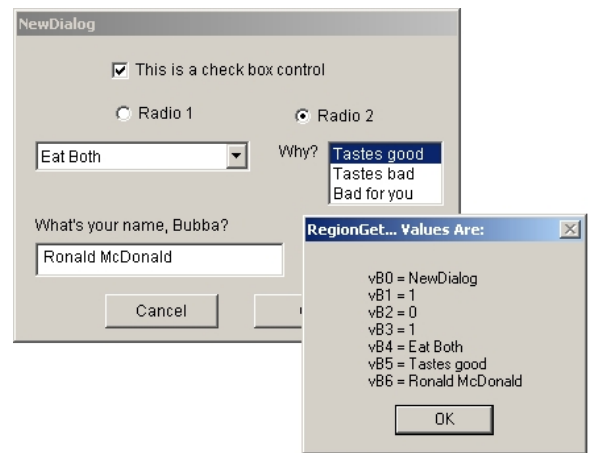
If you want to try this out for yourself, the macro code below can be copied into a WordPerfect document, as follows:

First, open a blank WordPerfect document and turn on your Macro Toolbar (Tools|Macros|Macro Toolbar).

Then, using Adobe's Text select feature, select the macro code below and press Ctrl+C to copy it to the Windows Clipboard.

Then, in the same blank WordPerfect document, press Ctrl+V to paste it into your document.

Last, click the Save & Compile Button on the Macro Toolbar, naming the macro what you want, e.g., Test.wcm. Although the dialog was originally made using the Dialog Editor, I've copied that dialog and pasted it into the macro for your use below (eliminating only the line numbers).



- **Sample RegionGet... Code**

```
//Begin macro code
Application (WordPerfect; "WordPerfect"; Default!; "EN") H=NTOC(OF90Ah)
Call(newDlg)
vQuit=False
DialogShow("new"; "WordPerfect"; cbnew)
Repeat Until(vQuit) DialogDismiss("new"; 1) DialogDestroy("new") Quit
  Label(cbnew) Switch(cbnew[3])
Caseof "OKBttn":
  vB0=RegionGetWindowText("new")
  vB1=RegionGetCheck("new.B1")
  vB2=RegionGetCheck("new.B2")
  vB3=RegionGetCheck("new.B3")
  vB4=RegionGetWindowText("new.B4")
  vB5=RegionGetSelectedText("new.B5")
  vB6=RegionGetWindowText("new.B6")
  MessageBox("RegionGet... Values Are: ";
    "vB0 = "+vB0+H+"vB1 = "+vB1+H+"vB2 = "+vB2+H+
    "vB3 = "+vB3+H+"vB4 = "+vB4+H+"vB5 = "+vB5+H+"vB6 = "+vB6)
Caseof "CancelBttn": vQuit=True
EndSwitch Return
  Label(newDlg)
DialogDefine (Dialog:"new"; Left:50; Top:60; Width:191; Height:122; Style:Percent!+NoCloseBox!;
Caption:"NewDialog")
DialogSetProperties (Dialog:"new"; FontName:"Arial"; FontSize:9p)
DialogAddCheckBox (Dialog:"new"; Control:"B1"; Left:41; Top:9; Width:108; Height:9; Text:"This is a
check box control"; MacroVar:var0; Style:CheckboxAuto!)
DialogAddRadioButton (Dialog:"new"; Control:"B2"; Left:42; Top:26; Width:55; Height:10;
ButtonText:"Radio 1"; MacroVar:var1; Style:RadioAuto!)
DialogAddRadioButton (Dialog:"new"; Control:"B3"; Left:120; Top:27; Width:55; Height:10;
ButtonText:"Radio 2"; MacroVar:var2; Style:RadioAuto!)
DialogAddComboBox (Dialog:"new"; Control:"B4"; Left:8; Top:42; Width:93; Height:56;
Style:Droplist!;MacroVar: var3)
  DialogAddListItem (Dialog:"new"; Control:"B4"; Item:"Eat Gravy";"Eat Biscuits";"Eat Both";"They'll Kill
Ya")
DialogAddListBox (Dialog:"new"; Control:"B5"; Left:135; Top:42; Width:50; Height:30; Style:0;
MacroVar:var4)
  DialogAddListItem (Dialog:"new"; Control:"B5"; Item:"Tastes good";"Tastes bad";"Bad for you")
DialogAddEditBox (Dialog:"new"; Control:"B6"; Left:8; Top:83; Width:108; Height:14;
Style:Left!+AutoHScroll!; MacroVar:var5; LimitText:100)
DialogAddPushButton (Dialog:"new"; Control:"OKBttn"; Left:103; Top:104; Width:49; Height:14;
Style:OKBttn!; ButtonText:"OK")
DialogAddPushButton (Dialog:"new"; Control:"CancelBttn"; Left:38; Top:104; Width:49; Height:14;
Style:CancelBttn!; ButtonText:"Cancel")
DialogAddText (Dialog:"new"; Control:"S1"; Left:8; Top:72; Width:93; Height:10; Style:Left!;
Text:"What's your name, Bubba?")
DialogAddText (Dialog:"new"; Control:"S2"; Left:107; Top:42; Width:24; Height:10; Style:Right!;
Text:"Why?")
Return
//End Macro Code
```

- **Other Region... Commands.** Just because a dialog has an initial given definition doesn't mean that it has to stay that way. You may want various controls to be displayed, or not, depending on other choices made in the dialog. You may want to predetermine the selection in a list depending on the selection in another, or you may want to reset a list entirely and set a different default selection. You can make a dialog be as fluid as you want during a callback. This section shows how to do a few such things.

- **RegionShowWindow.** Use **RegionShowWindow** to display, or not, various Controls within a dialog. In some contexts, you may want a Control to be seen and be available, but in other contexts, not.

The dialog name and the control within the dialog (case sensitive) is the 1st parameter, and the display value, Show! or Hide!, is the 2nd parameter. Use this code:

```
RegionShowWindow("vDialog.B1";Show!) to show a control and
RegionShowWindow("vDialog.B1";Hide!) to hide a control.
```

- **RegionResetList.** Use **RegionResetList** to clear all items from a List Box or Combo Box control, enabling you to add an entirely new list of items from which to select. Use this code:

```
RegionResetList("vDialog.B1")
```

- **RegionAddListItem.** Use **RegionAddListItem** to add one or more items to a List Box or Combo control. Use this code:

```
RegionAddListItem("vDialog.B1";"Apples") or RegionAddListItem("vDialog.B1";"Apples";"Pears")
```

- **RegionRemoveListItem.** Use **RegionRemoveListItem** to remove one item from a List Box or Combo Box control. Use this code:

```
RegionRemoveListItem("vDialog.B1";"Apples")
```

- **RegionSelectListItem.** Use **RegionSelectListItem** to select an item in a List Box or Combo Box control. It and can include an optional ending parameter, Select!, Unselect! or Extend!. Select! unselects all other selections and selects the item; Unselect! "unselects" the item and doesn't affect other selections; Extend! selects an item and adds it to any others which may already be selected. If you have used the **RegionResetList** command, described above, it's important that *some* item in the affected list be selected so that when a **RegionGetSelectedText** command occurs *something* will be gotten! Generally, use this code:

```
RegionSelectListItem("vDialog.B1";"Apples")
```

- **RegionSetCheck.** Use **RegionSetCheck** to set the value of a Check Box or Radio Button control box as checked or not. Use the Checked! or Unchecked!, as follows:

```
RegionSetCheck("vDialog.B1";Unchecked!) or RegionSetCheck("vDialog.B1";Checked!)
```

- **RegionSetFocus.** Use **RegionSetFocus** to cause the insertion point to be in a particular control in a dialog (or on the dialog itself). Possible uses: after a user clicks a non-ending Button control (one which does not end the callback) in a dialog which opens a non-looping dialog containing information or help), using this command resets the dialog's focus to where you want.

```
RegionSetFocus("vDialog.B1") [control] or RegionSetFocus("vDialog") [dialog itself]
```

- **RegionSetWindowText.** Use **RegionSetWindowText** to set or reset the text in a dialog's Caption Bar, Static Text or Edit Box control. For example, use of this command allows the text in a Static Text control box to change, depending on a user's selection in a List Box within the same dialog, so that if a List Box control allows a "Male" or "Female" selection, and the user selects, "Male", the text in Static Text control (above or beside an Edit Box control containing a person's name) would change to be, "Type the man's name", or "Type the woman's name"). Another example: use the command to set the initial text in an Edit Box control, e.g., a person's name (perhaps contained in a variable) or a specific date (e.g., "1/1/2003") which could be a string as just shown or a variable containing a string value. To change the dialog's Caption Bar, use the dialog's name as the 1st parameter. Here are examples:

```
RegionSetWindowText("vDialog.S1";"Type the woman's name:")
```

```
RegionSetWindowText("vDialog.B1";"John Doe") or RegionSetWindowText("vDialog.B1";vName)
```

```
RegionSetWindowText("vDialog";"New Text In Caption Bar")
```

- **Making a Double-Click End a Callback.** If you want a double-click on a List Box item to operate so that the loop ends, you can. It doesn't work so well with other controls, but you can try it and it might. The code varies slightly between WordPerfect 6.1 and higher WordPerfect versions. However, as a practical matter, since RegionGet... syntax changed beginning with WordPerfect 8.0, WordPerfect 6.1, 7.0, and 8.0 and higher, all need separate treatment. The double-click is picked up in either in the callback's array item [6] (WordPerfect 7.0 and higher) or in the callback's array item [7] (WordPerfect 6.1). My "callback paranoia statements" (some prior experience in WordPerfect 6.1 and/or 7.0 required

a 1 and 2 for an "OKBttn" and a "CancelBttn") for Caseof 1 and Caseof 2 are not used in the WordPerfect 8.0 and higher code, but are used in the WordPerfect 7.0 and 6.1 and cross-version code, below. Also, see J. Dan Broadhead's comments which follow this code.

WordPerfect 8.0 and higher:

```
Label(cbvDialog) Switch(cbvDialog[3])
    Caseof "OKBttn": var=RegionGetSelectedText("vDialog.B1") vLoop=True
    Caseof "CancelBttn": vQuit=True vLoop=True
    Caseof "B1": // this is the double-click routine:
        If ((cbvDialog[6] >> 16) = 2) var=RegionGetSelectedText("vDialog.B1") vLoop=True EndIf
    EndSwitch Return
```

WordPerfect 7.0 (my "callback paranoia statements" for Caseof 1 and Caseof 2 are used):

```
Label(cbvDialog) Switch(cbvDialog[3])
    Caseof "OKBttn": 1: RegionGetSelectedText(var;"vDialog.B1") vLoop=True
    Caseof "CancelBttn": 2: vQuit=True vLoop=True
    Caseof "B1": // this is the double-click routine:
        If ((cbvDialog[6] >> 16) = 2) RegionGetSelectedText(var;"vDialog.B1") vLoop=True EndIf
    EndSwitch Return
```

WordPerfect 6.1 (my "callback paranoia statements" for Caseof 1 and Caseof 2 are used):

```
Label(cbvDialog) Switch(cbvDialog[3])
    Caseof "OKBttn": 1: RegionGetSelectedText(var;"vDialog.B1") vLoop=True
    Caseof "CancelBttn": 2: vQuit=True vLoop=True
    Caseof "B1": // this is the double-click routine - notice that [7], not [6] is used
        If ((cbvDialog[7] >> 16) = 2) RegionGetSelectedText(var;"vDialog.B1") vLoop=True EndIf
    EndSwitch Return
```

Here are J. Dan Broadhead's comments:

Item [6] of the callback array is what is known as the "wparam". This is 1 of 2 values passed by Windows to the internal macro system. The other value is contained in [7], and is called the "lparam". For convenience, both the wparam ([6]) and the lparam ([7]) are broken into their hi and lo components, and are placed into elements [8] to [11] of the callback array as follows:

[8] = hi part of lparam

[9] = lo part of lparam

[10] = hi part of wparam

[11] = lo part of wparam

In the code "If ((cbvDialog[6] >> 16) = 2)", by using >> 16, you are really extracting the hi part of [6] (the wparam) by hand. You can access this value directly in element [10], so your code could be:

If (cbvDialog[10] = 2)

But elements [8] to [11] may not have been available in WP6, but it might have been (I'm not certain), so you might still have to use [6] >> 16.

But they are available in WP8.

And at least in WP8 and on, the following other controls provide access to a double-click notification (WP7 probably supported this as well, but I'm not sure about 6):

Combobox: [10] = 2

Pushbutton, Radiobutton, Checkbox: [10] = 5

Hotspot: [10] = 1

(in all cases, if [10] is not available in a version, then use [6] >> 16 as you did).

- **WordPerfect Version Control.** Recall that the syntax of RegionGet... commands is one thing in WordPerfect 6.1 and 7.0, and something else in WordPerfect 8.0 and higher. For example, `var=RegionGetSelectedText("vDialog.B1")` will not compile in WordPerfect 6.1; and, while `RegionGetSelectedText(var;"vDialog.B1")` will compile in WordPerfect 7.0 and higher, in WordPerfect 9.0, SP4 and higher, it will produce an obsolete warning message. Should you have a wish that your macro compile and run properly in WordPerfect 6.1 through 12.0, and that a double-click end the callback dialog without obsolete warning messages in Wp9.0, SP4 and higher, that can be accomplished using `IfPlatform - EndIfPlatform` statements, as follows (and thanks to J. Dan Broadhead for supplying the following code) (and, to be sure, you'd need similar `IfPlatform - EndIfPlatform` statements elsewhere in the macro if other non-cross-macro-version commands are present, as in Chapter 9's `wp9select.wcm`) – this only shows the double-click routine:

```
Label(cbvDialog) Switch(cbvDialog[3])
  Caseof 1;"OKBttn":
    IfPlatform (win!)
      RegionGetSelectedText(var;"vDialog.B1")
    EndIfPlatform
    IfPlatform (win32!)
      var=RegionGetSelectedText("vDialog.B1")
    EndIfPlatform
    vLoop=True
  Caseof 2;"CancelBttn": vQuit=True vLoop=True
  Caseof "B1":
    IfPlatform (win!)
      If ((cb50a[7]>>16)=2)
        RegionGetSelectedText(var;"vDialog.B1")
        vLoop=True
      EndIf
    EndIfPlatform
    IfPlatform (win32!)
      If ((cb50a[6]>>16)=2)
        var=RegionGetSelectedText("vDialog.B1")
        vLoop=True
      EndIf
    EndIfPlatform
  EndSwitch
Return
```

- **Commands At Beginning of DialogShow.** When displaying a dialog in the first place, you may want to modify what the dialog shows to exclude/include particular Controls within the dialog, or you may want to set the default text and/or particular Controls within the dialog, or you may want to set the default text and/or list items within various controls in the dialog. In my experience (despite what I've been told by others), it's best to include such commands AFTER and not BEFORE the DialogShow command, before the Repeat Until(vLoop) command. I've found such placement to work without fail, but I've found that using such commands BEFORE the DialogShow command doesn't always work. J. Dan Broadhead's comments, below, are far more illuminating than what I've just said, but with the qualification that I will likely start modifying my code to use a DialogLoad command as he suggests, I'm personally content with what I do.

So, what I do is: Place any such commands AFTER the DialogShow command but BEFORE the Repeat Until(vLoop) command. Some **excerpts** from Math.wcm, below, illustrate what I mean.

Having said that, and understanding that I'm once again totally exposing my ignorance (except as to what works for me), if this point is important to you, you should read what J. Dan Broadhead has to say (in response to what I've said above):

Before you access a dialog with any Region... commands or some other Dialog... commands, the dialog MUST exist. Its definition must have been loaded into memory in the macro system, AND it must have been created internal to Windows (whether it is visible or not).

For a Dialog Editor dialog, the dialog definition is not loaded into memory, and the dialog is not created in Windows, until you call DialogShow.

For a DialogDefine dialog, the definition exists in memory after the DialogDefine command (which created the definition in memory), but the dialog does not exist in Windows until DialogShow.

"Well, what's the difference?", I asked. He replied:

Actually, this depends on the version you are using. The above is the general rule, but it is slightly different in WP6. In WP6 (I believe) the definition of a DialogDefine dialog is created by DialogDefine, AND the dialog is created in Windows at that same time. Otherwise, I believe that the rules are the same.

The Region... commands access the actual dialog internal to Windows, not the dialog definition in memory. So you need to make sure the dialog actually exists in Windows before you can access it. This normally happens by calling DialogShow, but this command also displays it on the screen as well.

But you can force this to happen by calling DialogLoad. DialogShow calls DialogLoad first, and then it shows/displays the dialog. So if you call DialogLoad yourself, then the dialog will actually exist in Windows, even though it is hidden.

In WP8 on, if you try to access a dialog before it actually exists, then the macro system will call DialogLoad for you automatically. This may happen some or most of the time in WP7 as well, but there are probably some cases where it won't (inevitably the cases you need). And I believe in WP6, this would never happen automatically, but I could be wrong.

So to be safe, you can always call DialogLoad yourself.

BUT, there are really 2 different types of things you are talking about in this section. One of those involves manipulating the control itself (not its value, but the control), and the other involves the value of the control, and not the control itself.

These 2 things are distinct, and have clearly defined rules regarding their behavior.

The control itself can be manipulated at any time (after the control exists).

The contents of the control can only be changed AFTER DialogShow. And this can be tricky too depending on what version you are using.

Here's what happens.

When the dialog is first created in Windows, then a callback is sent to your callback label. This may not happen in WP6 or WP7 (I'm not sure, but I don't think so). Note that this doesn't happen if the dialog is created BEFORE you Call DialogShow. Remember from above, that a dialog is normally actually created when you call DialogShow. But if you call DialogLoad yourself, or if you call some command (like a Region... command) that causes DialogLoad to be called for you, and this is done BEFORE the DialogShow, then the macro system doesn't know what the callback label is for the dialog, and so the callback cannot actually be called.

Then, the controls are initialized with their "initial" values as defined in the Dialog Editor.

Then when you actually call DialogShow, the controls are set to the values of their associated variables. If an associated variable does not exist, then the value of the control is left alone, meaning that the initial value defined in the Dialog Editor is left in the control.

Then your callback is called again, to indicate that the controls have been pre-set. This may not happen in WP7 (I'm not sure), and does not happen in WP6.

Then the dialog is displayed on the screen, which causes the callback routine to be called again, to indicate it has been sized and displayed. This will not happen in WP6 and WP7.

Then your callback is called again, to indicate that it has received focus. This will not happen in WP6

and WP7.

So, here is the problem with trying to change the value of the control...

If you set the value fo the control BEFORE you call DialogShow, then those values will be over-written by the values of the associated variables (if any) when you call DialogShow, and your values you put in their before, will now be gone.

So this is why you have to wait till AFTER DialogShow to change the values of the controls.

The best way to change the values of the controls when you are about to display the dialog is to load the values that you want to display into the variables associated with the controls. What ever values are in the variables will be placed into controls every time you display the dialog.

If you can't or don't want to do this, you will either have to wait until after the DialogShow command, or you will have to watch for the special pre-set callback I mentioned above (not available in WP6, and might not be available in WP7).

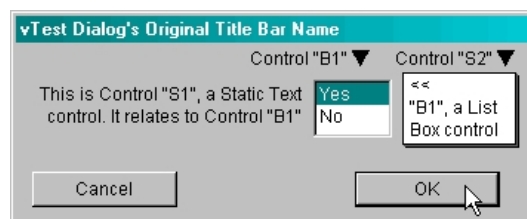
The reason that people suggest doing things before DialogShow, is to prevent the display of the dialog from changing values – it will display with the original values, and then it will change when you change the values yourself – possibly being visually disturbing to the user.

The same goes for hiding or showing controls on the dialog. If these are done when the dialog is still hidden (but after being created), then when it first shows, all the proper controls will be hidden or shown already by then.

! LEARNING BY EXAMPLE. I think the easiest way for this chapter to be easily understood is to present its content in a practical way by way of examples which illustrate what you need to know. Four examples will be considered: First (and simplest) example; Second Example (which builds upon the first); Third, and most complex example; and, Fourth, a simple but useful example you can use for messaging while a macro does something rather time consuming in the background. *Contrary to this manual's norm, links move forward, not to the chapter top, e.g., click on "First Example" to move to the "Second Example".*

FIRST EXAMPLE [click here to download the 1st & 2nd examples]

We'll start with this simple dialog and build upon it in the **Second Example**. The dialog's name is "vTest" and was made in the Dialog Editor. To be able to use "special characters" in the two top-right Static Text boxes (with the "▼"), the properties for each has been set as "WordPerfect" rather than "Standard". The List Box control, Control "B1" has been associated with variable vB1 in that control's properties. "OKBttn" and "CancelBttn" push buttons are also present.



THE CODE

```

1 Application (WordPerfect; "WordPerfect"; Default!; "EN")
2 vLoop=False
3 DialogShow("vTest"; "WordPerfect"; cbTest)
4 Repeat Until(vLoop)
5 If(Exists(vB1)) MsgText=vB1 Else MsgText="Variable vB1 does not exist" EndIf
6 MessageBox(;"vB1";MsgText)
7 If(x=2) Quit EndIf // could be Return instead of Quit
8 //[Do something else]
9 Return
10 // End of macro
11 Label(cbTest) Switch(cbTest[3])
12 Caseof "OKBttn";1: vClose
13 Caseof "CancelBttn";2: vClose
14 EndSwitch

```

```

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

```

HOW IT WORKS

- (1) Line 2: Variable vLoop is assigned a Boolean *False* value. Later, when variable vLoop is assigned a True value, the loop will stop.
- (2) Line 3: DialogShow() shows the dialog, and, since it is intended that a callback routine be used, I've named the called label as Label(cbTest), but it could be any other legal label name.
- (3) Line 4: The Repeat Until(vLoop) statement tells the dialog to remain visible until vLoop = *True*.
- (4) Lines 11-14: In the callback label, Label(cbTest), a Switch command switches amongst possible values of array [3] of that label. (Don't ask – I'm not smart enough to get into that!) I've used my "paranoia" statements described elsewhere to test for both the "OKBttn" (aka 1) and the "CancelBttn" (aka 2), but those numeric values are probably not necessary.
- (5) Lines 16-17: I've used a called label, Label(vClose) to dismiss the dialog. The value of the control ("OKBttn" or "CancelBttn" – or, at least theoretically, 1 or 2) is the second parameter of the DialogDismiss statement. If the value is "OKBttn" (or 1), variables (if any) contained in the dialog's controls are declared and assigned, but, if the value is "CancelBttn" (or 2), they are not.
- (6) Line 17: Variable x is assigned the value of MacroDialogResult. In the above example, that value will be 1 if the "OKBttn" was clicked or 2 if the "CancelBttn" was clicked.
- (7) Line 17: That label also contains an assignment of Boolean *True* to variable vLoop, thus ending the loop.
- (8) Lines 5-6: The macro then resumes its flow immediately after the Repeat Until (vLoop) statement, with a simple message box at Line 6. If the "OKBttn" push button control was used to terminate the loop, variable vB1 will have been declared and will have been assigned a string value of "Yes" or "No" in this example. However, if the "CancelBttn" push button control was used to terminate the loop, the value contained in Control "B1" will not be obtained and variable vB1 will not exist. The message box makes its report. Then, depending on the value of variable x (1 or 2), the macro will either stop (Quit) or continue with other activity.

PAST SINS REVISITED

Before learning better ways from J. Dan Broadhead, what I'd done (and, on occasion, may do until my old habits are fully displaced) was quite different than the above. I'd pretty much learned via the "monkey see, monkey do" approach, without much if any thought being given to the matter. As I said at the outset, "If it works, it works." However, in case you're doing stuff like I'd been doing, it is worthwhile to show you the "irregular" means I'd been using.

So, not so long ago, I'd have written the above code like this:

```

1 Application (WordPerfect; "WordPerfect"; Default!; "EN")
2 vLoop=False vQuit=False
3 DialogShow("vTest";"WordPerfect";cbTest)
4 Repeat Until(vLoop)
5 DialogDismiss("vTest";1) DialogDestroy("vTest")
6 If(Exists(vB1)) MsgText=vB1 Else MsgText="Variable vB1 does not exist" EndIf
7 MessageBox(;"vB1";MsgText)
8 If(vQuit) Quit EndIf // could be Return instead of Quit
9 //[Do something else]
10 Return
11 // End of macro
12     Label(cbTest) Switch(cbTest[3])
13 Caseof "OKBttn";1: vB1=RegionGetSelectedText("vTest.B1") vLoop=True
14 Caseof "CancelBttn";2: vQuit=True vLoop=True
15 EndSwitch
16 Return

```

PAST SINS REVISITED

Mind you, the above code does and has worked well for me, else I'd have stopped doing it a long time ago. That said, in the above dialog, IF the List Box control properties for Control "B1" REMAINS ASSOCIATED with variable vB1, variable vB1 *will be declared and assigned a value* EVEN IF the "CancelBttn" push button is clicked. Why? It's because the DialogDismiss("vTest";1) statement FORCES closing of the dialog with an effective MacroDialogResult of 1. I suppose that a thinking person would have figured that out a long long time ago, but, NO!

However, my practice was to NOT associate dialog controls with "permanent" (so to speak) variables, as will be discussed in a later example. A brief example of my "older" way is shown above in the vB1=RegionGetSelectedText("vTest.B1") statement. I'd apparently lived under the illusion (if I'd even thought about it) that such RegionGet... statements were necessary to assign variables values during callback routines. I'm not sure where I picked up that notion, but it had become habit. The underlying assumption I'd come to accept is not true at all, assuming (in the above example) that Control B1's control properties associate that control with variable vB1.

While instances may well be present to use one or many RegionGet... statements during a callback, they are certainly not as plentiful as I'd perceived. About my bad habit, J. Dan Broadhead was generous enough to say in his initial review of my handiwork, "If you rely on Region... commands to update the variables yourself, then I suppose there is little need to worry about dismissing the dialog, because you are not relying on what dismissing a dialog does for you. Since destroying a dialog will also remove it from the screen, which is all you want then anyway, then this does work for you. *But I suppose its like having a shovel, and then using the handle end to dig a hole. It works, but the shovel end was the end intended to be used, even though the handle end does work after a fashion!*" (Emphasis and the following comment are mine) Ha!

[Top of Chapter](#)
[Top of Example](#)
[Next Example](#)
SECOND EXAMPLE [[click here to download the 1st & 2nd examples](#)]

This example builds upon the [First Example](#). Dialog "vTest" now includes a third Push Button control (an "Other" Push Button I've named "ChangeBttn") and it includes a "Close" (x) button. Mostly, stuff has been added in the callback label which allows the dialog's appearance to change as various things occur. Have a look at dialog "vTest" in states of change as various things are clicked:

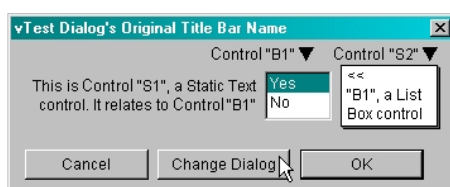


Figure 1



Figure 2

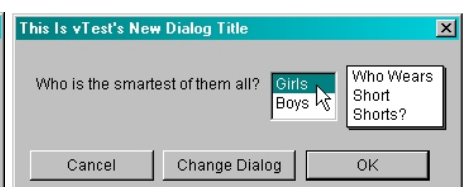


Figure 3



Figure 4



Figure 5



Figure 6

Above, Top row, left to right: Figure 1: Initial appearance, with the "ChangeBttn" about to be clicked; Figure 2: The "ChangeBttn" Push Button control has been clicked [notice that the top 2 Static Text controls containing ▼ are no longer visible, that the Title Bar's text, Control "S1"'s text, Control "S2"'s text is different, and that Control "B2"'s list has changed from "Yes" and "No" to "Girls" and "Boys" but that nothing is selected]; Figure 3: "Girls" is selected in Control "B1"'s List Box Control [Control "S2"'s text changed].

Above, Second row: Figure 4: "Boys" is selected in the same control [Control "S2"'s text changes again]; Figure 5: The ChangeBttn control is again clicked, followed by selecting "Yes" in the same List Box control [the Title

Bar is back the way it started as is Control "S1"'s Static Text control, the top 2 Static Text controls are visible, and Control "S2"'s text has changed; and Figure 6: "No" is selected in the List Box control [and the rest ditto Figure 5].

THE CODE

```

1 Application (WordPerfect; "WordPerfect"; Default!; "EN")
2 //A WM_SYSCOMMAND message is received when the user clicks Close from the system menu
3 box, double-clicks the system menu box, or presses Alt+F4; the value is 274
4 vLoop=False
5 DialogShow("vTest"; "WordPerfect"; cbTest)
6 OriginalTitle=RegionGetWindowText("vTest")
7 OriginalS1=RegionGetWindowText("vTest.S1")
8 OriginalS2=RegionGetWindowText("vTest.S2")
9 Repeat Until(vLoop)
10 MessageBox(;"x"; x)
11 If(Exists(vB1)) MsgText=vB1 Else MsgText="Variable vB1 does not exist" EndIf
12 MessageBox(;"vB1";MsgText)
13 Quit // End of macro
14   Label(cbTest)
15 If(cbTest[5]=274) cbTest[3]= 2 EndIf // If the Windows "Cancel" (x) button is clicked variable
16 cbTest[3] is assigned a value of 2, setting up that value in the following Switch statement; in the
17 assignment, ="CancelBttn" could have been used instead of =2
18 Switch(cbTest[3])
19 Caseof "OKBttn"; 1: vClose
20 Caseof "CancelBttn"; 2: vClose
21 Caseof "ChangeBttn": vReset
22 Caseof "B1": vPick
23 EndSwitch
24 Return
25   Label(vClose)
26 DialogDismiss("vTest";cbTest[3]) x=MacroDialogResult vLoop=True Return
27   Label(vReset)
28 var=RegionGetSelectedText("vTest.B1")
29 var=Substr(var; 1; 1)
30 RegionResetList("vTest.B1")
31 RegionSetWindowText("vTest.S2";"Select something in ""B1""")
32 Switch(var)
33   Caseof "Y"; "N":
34 RegionAddListItem("vTest.B1"; "Girls") RegionAddListItem("vTest.B1"; "Boys")
35 //RegionSelectListItem("vTest.B1"; "Girls") //ordinarily, you'd want to preselect an item
36 RegionSetWindowText("vTest"; "This Is vTest's New Dialog Title")
37 RegionSetWindowText("vTest.S1"; "Who is the smartest of them all?")
38 RegionShowWindow("vTest.S3"; Hide!) RegionShowWindow("vTest.S4"; Hide!)
39   Default:
40 RegionAddListItem("vTest.B1"; "Yes") RegionAddListItem("vTest.B1"; "No")
41 RegionSelectListItem("vTest.B1"; "Yes") //ordinarily, you'd want to preselect an item, as done
42 here
43 RegionSetWindowText("vTest"; OriginalTitle)
44 RegionSetWindowText("vTest.S1"; OriginalS1)
45 RegionShowWindow("vTest.S3"; Show!) RegionShowWindow("vTest.S4"; Show!)
46 EndSwitch
47 Return
48   Label(vPick)
49 var=RegionGetSelectedText("vTest.B1") var=Substr(var; 1; 1)
50 Switch(var)

```

```

51 Caseof "Y": vText="Yes! Yes! Yes!"
52 Caseof "N": vText="Bummer! I hate NO!"
53 Caseof "G": vText="Who Wears Short Shorts?"
54 Caseof "B": vText="Who's Your Daddy?"
55 EndSwitch
56 RegionSetWindowText("vTest.S2";vText)
57 Return

```

HOW EXAMPLE 2'S CALLBACK LABEL WORKS

These notes don't duplicate the parts already covered in [Example 1](#). Click that link for that. This discussion focuses on what's been added in the dialog and in the callback label.

- (1) Physical Changes in the dialog: Since this is a Macro Editor dialog, line references to these changes are non-existent; but, note that (a) A standard Window's Close (x) button is now present; and (b) an "Other" Push Button, named "ChangeBttn", is, also.
- (2) Lines 2-3: A new "comment" is present; this relates to information concerning the Windows "Close" (x) button, for your information. See Item (3)(c), below.
- (3) Lines 6-8: Below the DialogShow () command, three (3) variables are established (Original Title, Original S1, and Original S2). These variables will be used during the callback in Label(vReset); see item (5)(a), below.
- (4) In the callback label itself, 3 new lines have been added:
 - (a) Before the "Switch" statement, Lines 15-16: Line 15 contains a command which assigns a Windows "Close " (x) button a value which corresponds with the Switch() statement immediately following at Lines 18-23. The Result: If that button is clicked, variable cbTest[3] is assigned a value of 2 which makes it work within the Switch() statement at Lines 18-23, i.e., clicking the Windows Close button assigns that variable a value of 2, the same as a "CancelBttn", which will be responded to in the Switch statement.
 - In the Switch statement, Line 21: (b) Caseof "ChangeBttn": vReset; and (c) Line 22: Caseof "B1": vPick
- (5) Two additional labels have been added: (a) Label(vReset), Lines 27-47; and (b) Label(vPick), Lines 48-57.

Concerning these changes:

- (4)(a) Line 15 determines what happens if the standard Windows "Close" (x) button is clicked – variable cbTest[3] is assigned a value of 2 which is used in the Switch statement beginning at Line 18. So, the Windows "Close" button will be treated like a "CancelBttn", or 2, in the Switch.
- (4)(b) If the "ChangeBttn" is clicked, the contents of Label(vReset) (Item 5(a)) are called. Note that this has not been setup to end the callback, but is just used to change the dialog.

In Label(vReset), Line 28 creates (or redefines) variable var by getting the selected item in the Control "B1" List Box control by using the RegionGetSelectedText() command. Line 29 reassigns variable var to the 1st character of var, using the Substr() command.

Then, at Line 30, the Control "B1" List Box control is emptied using RegionResetList(). A Switch statement begins at Line 32 and ends at Line 46 which repopulates Control "B1"'s List Box control.

The 1st Caseof item instructs what happens if var="N" or "Y" (a substring of "No" or "Yes", respectively). In that event, Control "B1"'s list is populated with "Girls" and "Boys" using RegionAddListItem. Although I've commented out Line 35 (RegionSelectListItem()), I've merely done that to show you what happens if you DON'T determine a default selection – if that remains the case and a control is clicked which declares variable vB1 and closes the dialog, variable vB1 will have a value but it will be "", i.e., nothing. As well, Caseof "B1" (Line 21) will not work since

HOW EXAMPLE 2'S CALLBACK LABEL WORKS

variable var will be empty. Teaching point: Use RegionSelectListItem() when resetting a List Box's content. See Figure 2, above, which shows nothing selected in Control "B1". In the context of this example, since if variable var="N" or "Y", it is logically true that Control "S1" should be changed to be the alternatives present in this example. So, Line 36 changes the Dialog's Title Bar text and Line 37 changes Control "S1"'s text using RegionSetWindowText(), and Line 38 hides Controls "S3" and "S4" using RegionShowWindow(Hide!).

The Default switch element (Lines 39-45) does the reverse. Since var is not "N" or "Y", in the context presented, it has to be "G" or "B" (substrings of "Girls" and "Boys", respectively). Since no need exists to differentiate between them, and since they are the only other options intended, the Default switch element is all that is needed. Control "B1"'s List Box Control is again populated with "Yes" and "No", the original values (contained in variables OriginalTitle, OriginalS1, and OriginalS2, set at Lines 6 through 8 by RegionGetWindowText() commands), are set in the respective Static Text controls via RegionSetWindowText() commands, and Static Text controls "S3" and "S4" are made visible by RegionShowWindow() using Show! as the parameter.

- (4)(c) If the "B1" control is clicked, the contents of Label(vPick) (Item 5(b)) are called. Note that this has not been setup to end the callback, but is just used to change the dialog.

In Label(vPick): Line 49 creates (or redefines) variable var by getting the selected item in the Control "B1" List Box control by using the RegionGetSelectedText() command. Line 49 also reassigns variable var to the 1st character of var, using the Substr() command. A Switch statement at Lines 50-55 resets Control "S2" Static Text control as shown, depending on the value of variable var.

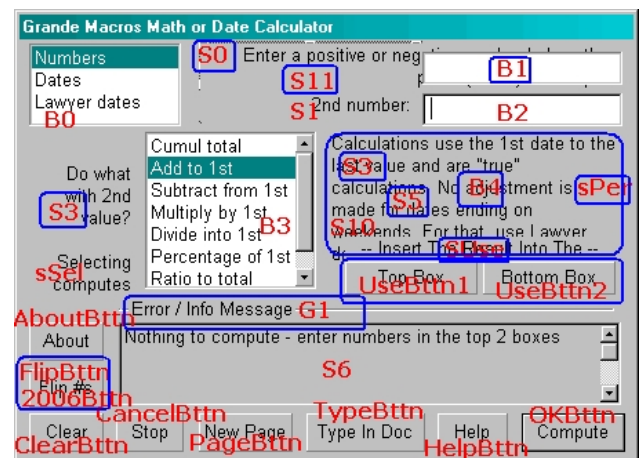
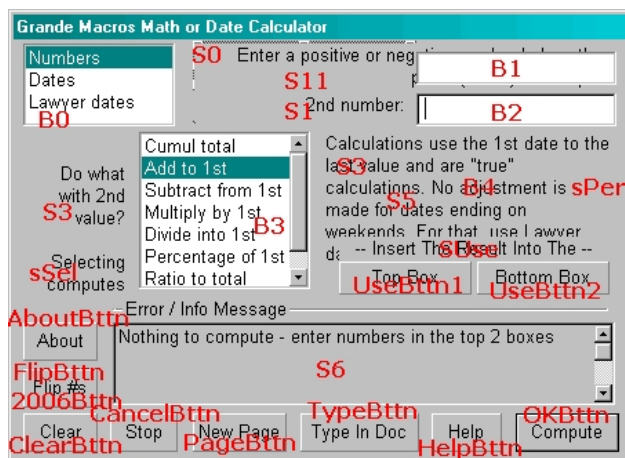
[Top of Chapter](#)

[Top of Example](#)

[Next Example](#)

THIRD EXAMPLE: EXTRACTS FROM MATH.WCM

A more elaborate example than the above is drawn from Math.wcm, presented in Chapter 9. Since THIS macro really doesn't intend to pass values in the dialog beyond the dialog itself (it is a math calculator which doesn't do anything outside itself), it relies upon numerous Region... commands to make it work. Examples of "Clean" variations of the main dialog were shown at the top of this chapter. Below are pictures showing Control Names, and their groupings. It's fussy to see as presented here, but look closely and it will hopefully make sense. These images are referenced in Chapter 6, [Specific Tasks](#).



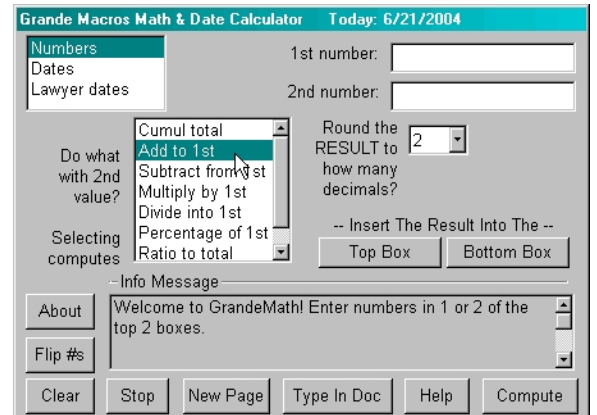
At left, red Control names are superimposed on the dialog – that's so in the right picture, also. However, the right picture also contains blue round rectangles around items which will change in their content, or not appear or appear, depending on what is selected in the callback label.

A "For What It's Worth" Note: Static Text Box controls: I find it convenient to identify Control Names with an "S" prefix, such as "S0", "S1", etc.; but, for controls in which computed values are located, I tend to identify those Controls with a "B" prefix, e.g., "B0", "B1", etc. Some consistent "naming" method will likely be helpful to you as you write your various macros.

Among other things, these pictures illustrate that different controls can occupy the same space – you'd want to include various RegionShowWindow commands to show or hide overlapping controls, else you'd have a big mess you'd have on your hands!

Associated commands before and after DialogShow take care of that initial cleanup and make the dialog look like that shown here. Now, it's not so messy!

In Math.wcm, the beginning code following comments at the top of the macro is shown below.



The Beginning Code

```

8 Application (WordPerfect; "WordPerfect"; Default!; "EN") H=NTOC(0F90Ah) OnError(vError)
9 QuickCorrectQuickBulletsSet(Off!) QuickCorrectQuickIndentSet(Off!) //added 6/4/2004
10 vSpecial=0 useSpecial=0 e=0
11 VarErrChk(Off!) // this turns off variable existence error checking
12 Display(On!) // this allows a user to "see" what's typed into a document, when the "Type" button is clicked
13 Today=?DateMonth+"/"+?DateDay+"/"+?DateYear
14 RegionSetWindowText("vMath"; "Grande Macros Math & Date Calculator" Today: "+Today")
15 vLoop=False DialogShow("vMath"; "WordPerfect"; cbMath)
16 tempN1="" tempN2="" tempL1=Today tempL2="0" tempC=""
17 tempD1=Today tempD2="0"
18 RegionSetWindowText("vMath.B1"; "")
19 RegionShowWindow("vMath.S10"; Hide!) RegionShowWindow("vMath.S11"; Hide!)
20 RegionShowWindow("vMath.2006Bbtn"; Hide!)
21 RegionShowWindow("vMath.sPer"; Hide!)
22 vMethod="N" // this sets the initial vMethod as "N" for "Numbers"
23 Call(setMethod) RegionSetWindowText("vMath.S6"; "Welcome to GrandeMath! Enter numbers in 1 or 2 of
24 the top 2 boxes.") RegionSetWindowText("vMath.G1"; "Info Message")
25 Repeat Until(vLoop) DialogDismiss("vMath"; 1) DialogDestroy("vMath") Quit

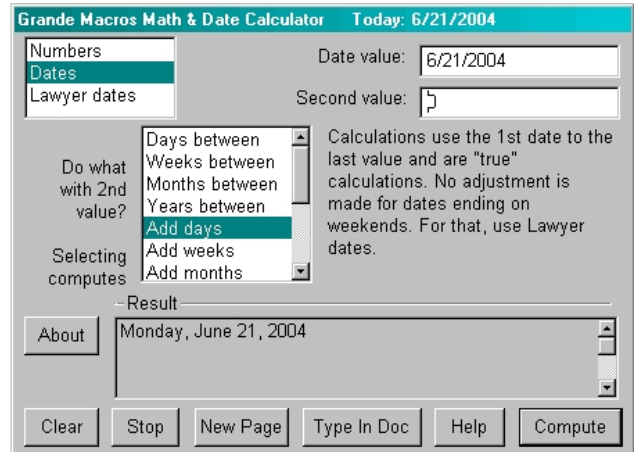
```

A good bit of that "clean up" was done by the commands located within Label(setMethod), called following the DialogShow() statement, above. The same Label(setMethod) is also called during the looping of the dialog and it does the rest, as is discussed below.

Commands Within the Callback Label. When a user clicks a particular control or makes some other selection, you may want the dialog to respond to the user's selections. As said above, Label(setMethod) was called at the outset of the dialog being displayed via DialogShow, and it is called while the loop is running.

For example, if a user selects "Dates" in Control B0, the dialog's appearance changes to look like this:

- S0 now reads "Date value:" instead of "1st number:";
- S1 now reads "Second value:" instead of "2nd number:";
- S3 no longer appears but S10, initially hidden, does;
- List Box items in B3 are totally different;
- The B4 Combo Box control doesn't appear;
- The message in S6 is different, as is the text in G1; and
- The FlipBtn, UseBtn1 and UseBtn2 Button controls no longer appear, nor does the sUse static control.



These changes occurred as a result of the combination of commands contained within Label(setMethod), the text of which is fully set forth below. As you read the code and look at the pictures on this and the prior page, you will be able to figure out what happened. Once again, this code is for WordPerfect 8.0 or higher.

Label(SetMethod)

```

126 Label(setMethod) // this routine shows/hides/resets some items when the upper left list box is clicked
127 var=RegionGetSelectedText("vMath.B0") vMethod=SubStr(var;1;1)
128 If(vMethod<>"N") vSpecial=0 EndIf
129 Switch(vMethod) // the user's selection will be Numbers, Dates, or Lawyer dates N, D, L
130   Caseof "N":
131     If(exists(saveT2)) //new on 6/4/2004: uses stored value when switching around between groups
132       RegionSetWindowText("vMath.B2";saveT2) Discard(saveT2) EndIf
133     RegionShowWindow("vMath.S10";Hide!) RegionShowWindow("vMath.S5";Show!)
134     RegionShowWindow("vMath.B4";Show!) RegionShowWindow("vMath.FlipBtn";Show!)
135     RegionShowWindow("vMath.2006Btn";Hide!)
136     RegionSetWindowText("vMath.S0";"1st number:")
137     RegionSetWindowText("vMath.S1";"2nd number:")
138     Nsel=RegionGetSelectedText("vMath.B3") vCalc=substr(Nsel;1;1)
139     Switch(Nsel)
140       Caseof "Cumul total":
141         saveT2=tempn2 //new on 6/4/2004: uses stored value when switching around between groups
142         RegionShowWindow("vMath.Ssel";Hide!)
143         RegionSetWindowText("vMath.S5";"Round ALL NUMBERS to how many decimals?")
144         RegionShowWindow("vMath.S3";Hide!)
145         RegionShowWindow("vMath.UseBtn1";Hide!) // new on 6/4/2004
146         RegionShowWindow("vMath.UseBtn2";Hide!) // new on 6/4/2004
147         RegionShowWindow("vMath.SUse";Hide!) // new on 6/4/2004
148         RegionShowWindow("vMath.B1";Hide!)
149         RegionShowWindow("vMath.S11";Show!) RegionShowWindow("vMath.S0";Hide!)
150         RegionShowWindow("vMath.FlipBtn";Hide!)
151         RegionSetWindowText("vMath.S1";"New number:")
152         RegionSetWindowText("vMath.B2";"") RegionSetFocus("vMath.B2")
153         If(exists(Ttxt2))
154           RegionSetWindowText("vMath.S6";Ttxt2)
155         Else
156           RegionSetWindowText("vMath.S6";"Nothing to compute - Select the decimal desired for this

```

```

157         session and then enter a number in the top box")
158         If(exists(vResult)) Discard(vResult) EndIf
159     EndIf
160     Default:
161         RegionShowWindow("vMath.Ssel"; Show!)
162         RegionSetWindowText("vMath.S5"; "Round the RESULT to how many decimals?")
163         RegionShowWindow("vMath.S3"; Show!)
164         RegionShowWindow("vMath.UseBbtn1"; Show!) // new on 6/4/2004
165         RegionShowWindow("vMath.UseBbtn2"; Show!) // new on 6/4/2004
166         RegionShowWindow("vMath.SUse"; Show!) // new on 6/4/2004
167         RegionShowWindow("vMath.FlipBbtn"; Show!)
168         RegionShowWindow("vMath.B1"; Show!)
169         RegionShowWindow("vMath.S11"; Hide!) RegionShowWindow("vMath.S0"; Show!)
170         RegionSetWindowText("vMath.S0"; "1st number: ")
171         RegionSetWindowText("vMath.S1"; "2nd number: ") RegionSetFocus("vMath.B1")
172         RegionSetWindowText("vMath.S6"; "")
173         tempN1=RegionGetWindowText("vMath.B1")
174         tempN2=RegionGetWindowText("vMath.B2")
175     EndSwitch
176     If(Nsel="Ratio to total" or Nsel="Percentage of 1st")
177         RegionShowWindow("vMath.sPer"; Show!)
178     Else
179         RegionShowWindow("vMath.sPer"; Hide!)
180     EndIf
181     Caseof "D": //revised 6/4/2004
182         RegionShowWindow("vMath.sPer"; Hide!)
183         RegionShowWindow("vMath.S10"; Show!) RegionShowWindow("vMath.S5"; Hide!)
184         RegionShowWindow("vMath.2006Bbtn"; Hide!)
185         RegionShowWindow("vMath.UseBbtn1"; Hide!) // new on 6/4/2004
186         RegionShowWindow("vMath.UseBbtn2"; Hide!) // new on 6/4/2004
187         RegionShowWindow("vMath.SUse"; Hide!) // new on 6/4/2004
188         RegionShowWindow("vMath.B1"; Show!)
189         RegionShowWindow("vMath.S11"; Hide!) RegionShowWindow("vMath.S0"; Show!)
190         RegionShowWindow("vMath.B4"; Hide!) RegionShowWindow("vMath.FlipBbtn"; Hide!)
191         RegionShowWindow("vMath.2006Bbtn"; Hide!)
192         RegionSetWindowText("vMath.S10"; "Calculations use the 1st date to the last value and are ""true""
193         calculations. No adjustment is made for dates ending on weekends. For that, use Lawyer dates.")
194         RegionSetWindowText("vMath.S0"; "Date value: ") var=RegionGetSelectedText("vMath.B3")
195         var=substr(var; 1; 1)
196         If(var="D" or var="W" or var="M" or var="Y")
197             RegionSetWindowText("vMath.S1"; "Date value: ")
198         Else
199             RegionSetWindowText("vMath.S1"; "Second value: ")
200         EndIf
201         Dsel=RegionGetSelectedText("vMath.B3") x=substr(Dsel; 1; 1)
202         tempD2=RegionGetWindowText("vMath.B2")
203         y=strpos(tempD2; "/")
204         Switch(x)
205             Caseof "D"; "W"; "M"; "Y":
206                 If(y>0)
207                     RegionSetWindowText("vMath.B2"; tempD2) Else
208                     RegionSetWindowText("vMath.B2"; "")
209                 EndIf
210             Default:
211                 If(y>0)
212                     RegionSetWindowText("vMath.B2"; "0") Else
213                     RegionSetWindowText("vMath.B2"; tempD2)
214                 EndIf
215             EndSwitch
216         RegionSetFocus("vMath.B2")
217     Caseof "L":

```



```

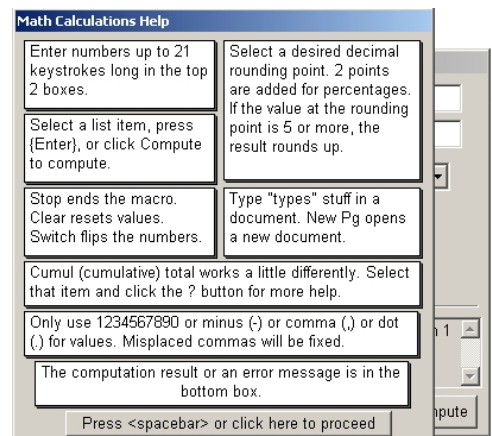
218 RegionShowWindow("vMath.sPer"; Hide!)
219 RegionShowWindow("vMath.S10"; Show!) RegionShowWindow("vMath.S5"; Hide!)
220 tempL1=RegionGetWindowText("vMath.B1")
221 tempL2=RegionGetWindowText("vMath.B2")
222 RegionShowWindow("vMath.2006Btn"; Show!)
223 RegionShowWindow("vMath.UseBtn1"; Hide!) // new on 6/4/2004
224 RegionShowWindow("vMath.UseBtn2"; Hide!) // new on 6/4/2004
225 RegionShowWindow("vMath.SUse"; Hide!) // new on 6/4/2004
226 RegionShowWindow("vMath.S10"; Show!)
227 RegionShowWindow("vMath.B1"; Show!)
228 RegionShowWindow("vMath.S11"; Hide!) RegionShowWindow("vMath.S0"; Show!)
229 RegionShowWindow("vMath.B4"; Hide!) RegionShowWindow("vMath.FlipBtn"; Hide!)
230 RegionShowWindow("vMath.2006Btn"; Show!)
231 RegionSetWindowText("vMath.S10"; "The date is day 'zero' for computing purposes. If an end date
232 falls on a weekend or legal holiday, the end date is usually the next business day. 12 OS §2006.
233 Weekends are adjusted.") RegionSetWindowText("vMath.S0"; "Date value: ")
234 RegionSetWindowText("vMath.S1"; "Number value: ")
235 Lsel=RegionGetSelectedText("vMath.B3")
236 EndSwitch
237 Return

```

I understand that the above may be obtuse, and, if you're like me, it helps to "see" things in action to get a better understanding. Run the macro and you'll have a better feel for what's going on here. [Click here to download the macro](#) and run it for yourself.

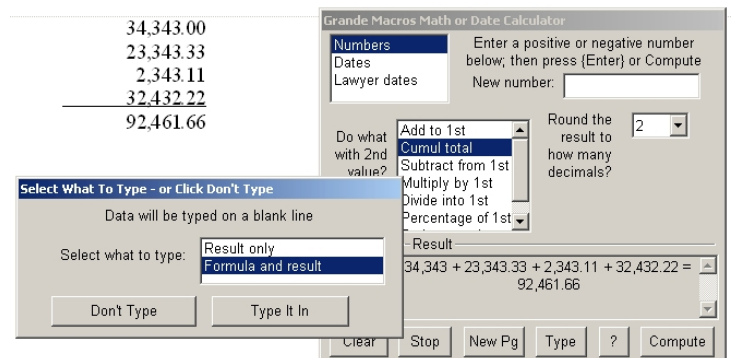
- **Using DialogShow During Callbacks.** While you can't use another DialogShow command which contains a callback (while one such dialog is looping – but see [J. Dan Broadhead's comments](#), above, as to later WordPerfect versions), you can use a DialogShow command which doesn't contain a callback during that loop. While MessageBox commands could also be used, they are less flexible and aren't all that pretty to look at. Here are 2 examples from [Math.wcm](#), below.

- **Information Dialogs.** This usage is really just a glorified MessageBox, except that it's a dialog called by DialogShow. In this picture, a user has clicked the "?" button in the Math.wcm dialog, the Control name being "HelpBtn".



The PushButton control in this dialog is really an "OKBtn" but with different text. The dialog which appears varies depending on the value of the variable "gotten" from Region "vMath.B0" (Numbers, Dates, Lawyer Dates).

- **Dialogs That Can Do Something.** While such dialogs certainly do not have the flexibility of a dialog which uses a callback routine, still, a dialog without a callback can be used to cause the macro to "do" something while the main dialog (with the callback) is running. Instead of using RegionGet... commands (usable only with callback dialogs), make dialogs which assign a variable to a particular control in the dialog, and use the MacroDialogResult command as needed. That will be explained shortly. But, for now, have a look at the above dialog which opens when the "Type" (control "TypeBtn") is clicked in Math.wcm. In the macro excerpt below, the *Caseof* "TypeBtn" lines show how this happened.



Commands used in Math.wcm which use many of the commands described in this chapter are illustrated in excerpts from that macro, with numerous *// comments like this*, below. While not all called routines

appear in the following excerpts, all of them are in Chapter 9, `Math.wcm`, below.

```

21     Label(cbMath) Switch(cbMath[3]) // this callback label does the work of the macro
22     Caseof "B0": // clicking a control causes action: B0 is the primary list (Numbers, Dates, etc.)
23         resetList setMethod
24         If(vCalc<>"C")
25             SwitchLists
26             If(Exists(q))
27                 Discard(q)
28             EndIf
29         EndIf
30     Caseof "B3": // control B3 is the middle list box
31         setMethod
32         If(vCalc<>"C")
33             SwitchLists
34             If(Exists(q))
35                 Discard(q)
36             EndIf
37         EndIf
38     Caseof "B4": // control B4 is the decimal combo box
39         setMethod
40         If(vCalc<>"C")
41             SwitchLists
42             If(Exists(q))
43                 Discard(q)
44             EndIf
45         EndIf
46         RegionSetFocus("vMath.B1")
47     Caseof "OKBttn": // control OKBttn is the "Compute" button
48         SwitchLists
49         If(Exists(q))
50             Discard(q)
51         Else
52             If(vCalc="C")
53                 RegionSetFocus("vMath.B2")
54             Else
55                 RegionSetFocus("vMath.B1")
56             EndIf
57         EndIf
58     Caseof "TypeBttn": // the Type button
59         If(Exists(vResult))
60             vType
61         Else
62             RegionSetWindowText("vMath.S6"; "Nothing to type - enter numbers in any needed boxes")
63         EndIf
64         RegionSetFocus("vMath.B1")
65     Caseof "CancelBttn": vLoop=True // the "Stop" button
66     Caseof "HelpBttn": // the "Help" button rev 6/4/2004 to deal with scientific notation, adding a new "help" dialog
67         Switch(vMethod)
68             Caseof "N": x=RegionGetSelectedText("vMath.B3") x=substr(x;1;1)
69             Switch(x)
70                 Caseof "C": var="vHelp7"
71                 Default:
72                     If(vSpecial<2)
73                         var="vHelp"
74                     Else
75                         var="vHelp8"
76                     EndIf
77             EndSwitch
78         Caseof "D": var="vHelp2"
79         Caseof "L": var="vHelp3"
80     EndSwitch

```

```

81 // var (above) is the name of the dialog to open, depending on the value of variable vMethod
82 DialogShow(var; "vMath") RegionSetFocus("vMath.B1")
83 If(vCalc="C")
84     RegionSetFocus("vMath.B2")
85 Else
86     RegionSetFocus("vMath.B1")
87 EndIf
88 Caseof "AboutBttn": // the "About" button
89     DialogShow("vHelp4"; "vMath")
90     If(vCalc="C")
91         RegionSetFocus("vMath.B2")
92     Else RegionSetFocus("vMath.B1")
93 EndIf
94 Caseof "2006Bttn": DialogShow("vHelp6"; "vMath") RegionSetFocus("vMath.B1")
95 Caseof "ClearBttn": // the Clear button
96     setMethod vSet
97     If(exists(q))
98         Discard(q)
99     EndIf
100     If(vCalc="C")
101         RegionSetFocus("vMath.B2")
102     Else
103         RegionSetFocus("vMath.B1")
104     EndIf
105 Caseof "FlipBttn": // the "Flip #s" button
106     vReverse SwitchLists
107     If(exists(q))
108         Discard(q)
109     EndIf
110     RegionSetFocus("vMath.B1")
111 Caseof "PageBttn": // the "New Page" button
112     OnError Call(vError) FileNew OnError
113     If(vCalc="C")
114         RegionSetFocus("vMath.B2")
115     Else
116         RegionSetFocus("vMath.B1")
117     EndIf
118 Caseof "UseBttn1": var=1 setUse // new on 6/4/2004: inserts result in top box
119 Caseof "UseBttn2": var=2 setUse // new on 6/4/2004: inserts result in bottom box
120 EndSwitch Return

```

At Line 60, above, Label(vType) was called (conditioned upon the existence of variable vResult). Much of the associated code is shown below.

```

950 Label(vWrite) // this label is called from Label(vType); below: it insures that decimal values are handled well
951 w=strPos(var; ".")
952 Switch(w)
953     Caseof 0:
954         If(varD<>0)
955             var=var+"." y=0
956             Repeat
957                 var=var+"0" y=y+1
958             Until(y=varD)
959         EndIf
960     Default: y=substr(var; w+1; strlen(var)) z=strlen(y)
961     If(z<varD)
962         Repeat
963             var=var+"0" z=z+1
964         Until(z=varD)
965     Else
966         vcomma var=strnum(var) // Label(vComma) is called and the result is used in var=strnum(var)

```

```

967     If(varD=0)
968         var=Integer(roundoff(var; 1/(10**varD))) newnum Else
969         var=numstr(var) x=strPos(var; ".")
970         If(x>0)
971             var=var+"000001"
972         EndIf
973         var=strnum(var)
974         var = RoundOff (var; (1/(10**varD))) newnum
975     EndIf
976 EndIf
977 EndSwitch
978 Return
979 Label(vType) // this routine types stuff into the open document
980 If(vMethod="Q")
981     vdial="vType2" Else vdial="vType"
982 EndIf
983 DialogShow(vdial) x=MacroDialogResult
984 If(x=2)
985     Return
986 EndIf
987 TypeWhat=Substr(TypeWhat; 1; 1)
988 x=?DocBlank
989 If(x=false)
990     PosLineEnd HardReturn
991 EndIf
992 varD=RegionGetSelectedText("vMath.B4") varD=strnum(varD)
993 If(TypeWhat="R")
994     Switch(vMethod)
995         Caseof "N":
996             If(vCalc="C")
997                 var=yresult newnum
998                 Type(var) RegionSetFocus("vMath.B2")
999                 Else Type(vResult) RegionSetFocus("vMath.B1")
1000             EndIf
1001             Default: Type(vResult)
1002         EndSwitch
1003     Return
1004 EndIf
1005 Switch(vMethod)
1006     Caseof "N": // what to type if numbers are being computed
1007         Switch(vCalc)
1008             Caseof "A": Type(FirstN+ " + "+SecondN+ " = "+vResult+ " [rounded to "+varD+ " decimals]")
1009             Caseof "S": Type(FirstN+ " * "+SecondN+ " = "+vResult+ " [rounded to "+varD+ " decimals]")
1010             Caseof "M": Type(FirstN+ " x "+SecondN+ " = "+vResult+ " [rounded to "+varD+ " decimals]")
1011             Caseof "D": Type(FirstN+ " ÷ "+SecondN+ " = "+vResult+ " [rounded to "+varD+ " decimals]")
1012             Caseof "P": Type(SecondN+ " ÷ "+FirstN+ " = "+vResult+ " [rounded to "+varD+ " decimals]")
1013             Caseof "R": Type(SecondN+ " ÷ (" +FirstN+ " + "+SecondN+ ") = "+vResult+ " [rounded to
1014                 "+varD+ " decimals]")
1015             Caseof "C": n=1 UnderlineTabs (Yes!) UnderlineSpaces (Yes!)
1016                 Repeat
1017                     var=T[n] Call(vWrite) typevar=var
1018                     Tab Tab TabDecimal Type(typevar) HardReturn n=n+1
1019                 Until (T[n]="")
1020                 var=vResult vWrite
1021                 PosLineUp PosLineBeg PosCharNext SelectMode(On!) PosLineEnd
1022                 AttributeAppearanceToggle(Underline!) SelectMode(Off!) PosLineDown
1023                 Tab Tab TabDecimal Type(var) HardReturn
1024             EndSwitch
1025         Caseof "D": // what to type if regular dates are being calculated
1026             If(SecondN=1)
1027                 plural="" Else plural="s"

```

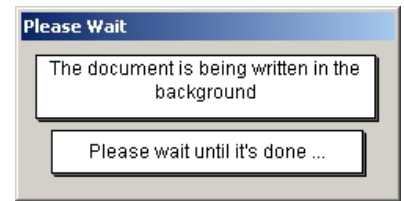
```

1028 EndIf
1029 Switch(vCalc)
1030   Caseof 1: Type("Days between "+SecondN+", and "+FirstN+" = "+vResult)
1031   Caseof 2: Type("Weeks between "+SecondN+", and "+FirstN+" = "+vResult)
1032   Caseof 3: Type("Months between "+SecondN+", and "+FirstN+" = "+vResult)
1033   Caseof 4: Type("Years between "+SecondN+", and "+FirstN+" = "+vResult)
1034   Caseof 5: Type(FirstN+" + "+SecondN+" day"+plural+" = "+vResult)
1035   Caseof 6: Type(FirstN+" + "+SecondN+" week"+plural+" = "+vResult)
1036   Caseof 7: Type(FirstN+" + "+SecondN+" month"+plural+" = "+vResult)
1037   Caseof 8: Type(FirstN+" + "+SecondN+" year"+plural+" = "+vResult)
1038   Caseof 9: Type(FirstN+" "+SecondN+" day"+plural+" = "+vResult)
1039   Caseof 10: Type(FirstN+" "+SecondN+" week"+plural+" = "+vResult)
1040   Caseof 11: Type(FirstN+" "+SecondN+" month"+plural+" = "+vResult)
1041   Caseof 12: Type(FirstN+" "+SecondN+" year"+plural+" = "+vResult)
1042 EndSwitch
1043 Caseof "L": // what to type if Lawyer dates are being calculated
1044   If(SecondN=1)
1045     plural="" Else plural="s"
1046   EndIf
1047   Switch(vCalc)
1048     Caseof 1: Type(FirstN+" + "+SecondN+" day"+plural+" = "+vResult)
1049     Caseof 2: Type(FirstN+" + "+SecondN+" week"+plural+" = "+vResult)
1050     Caseof 3: Type(FirstN+" + "+SecondN+" month"+plural+" = "+vResult)
1051     Caseof 4: Type(FirstN+" + "+SecondN+" year"+plural+" = "+vResult)
1052   EndSwitch
1053 EndSwitch
1054 RegionSetFocus("vMath.B2")
1055 Return
1056 Label(vComma) // this strips commas in strings which produce erroneous values in NumStr
1057 x=strpos(var; ",")
1058 If(x>0)
1059   While (x>0)
1060     y=substr(var; 1;x-1)+substr(var; x+1; strlen(var)) var=y x=strpos(var; ",")
1061   EndWhile
1062 EndIf
1063 Return

```

EXAMPLE 4: WAITMESSAGE

I'll close this chapter with a handy example of a "Wait Message" which will appear to let users know that a document is being written in the background, and that they should wait until it's done. The macro can be modified to suit your needs. Although the dialog used was originally made by using the Macro Editor, I've pasted the dialog into the macro as code so that you can copy and paste it, "as is", if you want. This code will work in WordPerfect 8.0 and higher but it will need to be modified for use in WordPerfect 6.1 and 7.0.



When run, the code produces a dialog that looks like the above picture, and the dialog is closed and destroyed when the Call(KillMsg) command is encountered in the macro.

```
vMsg1="The document is being written in the background"
vMsg2="Please wait until it's done ..."
Call(WaitDlg)
DialogShow("vWait"; "WordPerfect"; cbWait)
Wait(30)
// this above Wait statement is just there for demonstration purposes; for practical use, use the Call(KillMsg) line at the
// point in your macro that you want the dialog to be cancelled
Call(KillMsg)
Quit // instead of the Quit command, use whatever other macro flow commands you prefer, once the dialog is closed
Label(cbWait) // this is the callback label
If(cbWait[3]="CancelBttn")
    Assert(CancelCondition!)
EndIf
Return
Label(KillMsg) // this label's contents assert a cancel condition and destroy the dialog
DialogDismiss("vWait"; 2) Return
Label(WaitDlg) // in this label, "vWait" dialog is defined; you could make it in the Macro Editor
DialogDefine (Dialog:"vWait"; Left:50; Top:60; Width:136; Height:52;
Style:Sizeable!+Percent!+NoCloseBox!; Caption:"Please Wait")
DialogSetProperties (Dialog:"vWait"; FontName:"Arial"; FontSize:9p)
DialogAddText (Dialog:"vWait"; Control:"Static1"; Left:5; Top:4; Width:125; Height:23;
Style:ShadowBox!+Center!; Text:vMsg1)
DialogAddPushButton (Dialog:"vWait"; Control:"CancelBttn"; Left:3; Top:3; Width:0; Height:0;
Style:CancelBttn!; ButtonText:"Cancel")
DialogAddText (Dialog:"vWait"; Control:"Static2"; Left:11; Top:30; Width:115; Height:16;
Style:ShadowBox!+Center!; Text:vMsg2)
Return
```

NOTES

NOTES

NOTES

Chapter 9

Macro Examples

Links: The **Chapter Name**, above, moves to the next chapter. 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.

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
Macros In This Chapter				
Math.wcm		ConvertFE.wcm	Wp9select.wcm	

The purpose of this chapter is to present you with fully developed macros which utilize many if not most of the routines described in Chapters 3-8, one way or another. Aside from various **// comments** like this included in the macros, little expository information is provided. For that, read the parts of this manual which relate to the techniques used. In this manual's release, both macros below were substantially modified from their former versions. Many, not all, changes are shown with **//green comments**.

Dialogs in these macros were made in WordPerfect's Macro Dialog Editor. So, you won't see any commands which create the various dialogs. You can download these macros at my website as shown below and then you will have the full macro(s), including all their dialogs. Wp9select.exe works in WordPerfect 6.1 or later, but the other two work only in WordPerfect 8.0 or later.

The macros should be seen as illustrating some of the various means available to accomplish what they do – I'm sure that each can be made "better", and, doubtless, I'll be making changes in them as I find better ways to write macros – or fix any problems which may exist in the macros in their present form, just as I've done in the June 2004 version. You shouldn't see the code used as "limiting" or "exclusive" – other, perhaps better, means exist to do what these macros do.

Math.wcm is a number and date calculator which performs various types of computations while running WordPerfect and which can insert calculated results or formula and results into a WordPerfect document. Pictures of some Math.wcm dialogs are shown in Chapter 9.

ConvertFE.wcm converts either endnotes or footnotes to plain text at the end of a document, having the appearance of endnotes, but they really aren't. It is useful in preparing a document for a publication of some type in which "real" endnotes/footnotes are not desirable, as in a journal article. The primary dialog in ConvertFE.wcm looks like the picture here. Other dialog illustrations are in **Chapter 5**, above.

Wp9select.wcm turns on WordPerfect 9's text selection method if it isn't already on when run in WordPerfect 10 through 12.

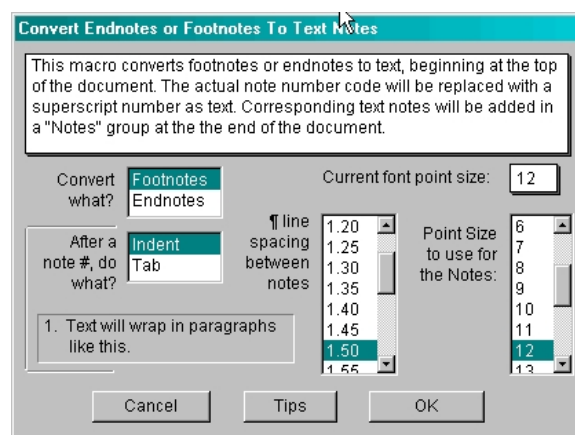
Each macro is downloadable in a self-extracting Winzip.exe file, as follows:

Math.wcm: <http://www.dogloudenback.com/wp/Math.exe>

CovertFE.wcm: <http://www.dogloudenback.com/wp/ConvertFE.exe>

Wp9select.wcm: <http://www.dogloudenback.com/wp/wp9select.exe>

Feel free to use and edit these macros as you want. Please DO NOT FEEL FREE to post the macros on any website without my prior written consent or to sell the macros to anyone.



```
1 //MATH.WCM
2 //Math.wcm © 2002-2004 by Doug Loudonback, Oklahoma City, All Rights Reserved
3 //A part of v5.5 and v6.0 Grande Macros, but without the QDRO date calculator element in the lawyer
4 commercial version of the Grande Macros program
5 //This macro may be distributed or edited as you want, but it may not be sold or posted
6 on any website without my express written permission.
7 //Revision Date: June 4, 2004
8 Application (WordPerfect; "WordPerfect"; Default!; "EN") H=NTOC(0F90Ah) OnError(vError)
9 QuickCorrectQuickBulletsSet(Off!) QuickCorrectQuickIndentSet(Off!) //added 6/4/2004
10 vSpecial=0 useSpecial=0 e=0
11 VarErrChk(Off!) // this turns off variable existence error checking
12 Display(On!) // this allows a user to "see" what's typed into a document, when the "Type" button is clicked
13 Today=?DateMonth+"/"+?DateDay+"/"+?DateYear
14 RegionSetWindowText("vMath"; "Grande Macros Math & Date Calculator Today: "+Today)
15 vLoop=False DialogShow("vMath"; "WordPerfect"; cbMath)
16 tempN1="" tempN2="" tempL1=Today tempL2="0" tempC=""
17 tempD1=Today tempD2="0"
18 RegionSetWindowText("vMath.B1"; "")
19 RegionShowWindow("vMath.S10"; Hide!) RegionShowWindow("vMath.S11"; Hide!)
20 RegionShowWindow("vMath.2006Bttn"; Hide!)
21 RegionShowWindow("vMath.sPer"; Hide!)
22 vMethod="N" // this sets the initial vMethod as "N" for "Numbers"
23 Call(setMethod) RegionSetWindowText("vMath.S6"; "Welcome to GrandeMath! Enter numbers in
24 1 or 2 of the top 2 boxes.") RegionSetWindowText("vMath.G1"; "Info Message")
25 Repeat Until(vLoop) DialogDismiss("vMath"; 1) DialogDestroy("vMath") Quit
26 Label(cbMath) Switch(cbMath[3]) // this callback label does the work of the macro
27 Caseof "B0": // clicking a control causes action; B0 is the primary list (Numbers, Dates, etc.)
28     resetList setMethod
29     If(vCalc<>"C")
30         SwitchLists
31         If(Exists(q))
32             Discard(q)
33         EndIf
34     EndIf
35 Caseof "B3": // control B3 is the middle list box
36     setMethod
37     If(vCalc<>"C")
38         SwitchLists
39         If(Exists(q))
40             Discard(q)
41         EndIf
42     EndIf
43 Caseof "B4": // control B4 is the decimal combo box
44     setMethod
45     If(vCalc<>"C")
46         SwitchLists
47         If(Exists(q))
48             Discard(q)
49         EndIf
50     EndIf
51     RegionSetFocus("vMath.B1")
52 Caseof "OKBttn": // control OKBttn is the "Compute" button
53     SwitchLists
54     If(Exists(q))
55         Discard(q)
56     Else
```

```
57     If(vCalc="C")
58         RegionSetFocus("vMath.B2")
59     Else
60         RegionSetFocus("vMath.B1")
61     EndIf
62 EndIf
63 Caseof "TypeBttn": // the Type button
64     If(exists(vResult))
65         vType
66     Else
67         RegionSetWindowText("vMath.S6"; "Nothing to type - enter numbers in any needed boxes")
68     EndIf
69     RegionSetFocus("vMath.B1")
70 Caseof "CancelBttn": vLoop=True // the "Stop" button
71 Caseof "HelpBttn": // the "Help" button rev 6/4/2004 to deal with scientific notation, adding a new "help" dialog
72     Switch(vMethod)
73         Caseof "N": x=RegionGetSelectedText("vMath.B3") x=substr(x; 1; 1)
74         Switch(x)
75             Caseof "C": var="vHelp7"
76             Default:
77                 If(vSpecial< 2)
78                     var="vHelp"
79                 Else
80                     var="vHelp8"
81                 EndIf
82             EndSwitch
83         Caseof "D": var="vHelp2"
84         Caseof "L": var="vHelp3"
85     EndSwitch
86     // var (above) is the name of the dialog to open, depending on the value of variable vMethod
87     DialogShow(var; "vMath") RegionSetFocus("vMath.B1")
88     If(vCalc="C")
89         RegionSetFocus("vMath.B2")
90     Else
91         RegionSetFocus("vMath.B1")
92     EndIf
93 Caseof "AboutBttn": // the "About" button
94     DialogShow("vHelp4"; "vMath")
95     If(vCalc="C")
96         RegionSetFocus("vMath.B2")
97     Else RegionSetFocus("vMath.B1")
98     EndIf
99 Caseof "2006Bttn": DialogShow("vHelp6"; "vMath") RegionSetFocus("vMath.B1")
100 Caseof "ClearBttn": // the Clear button
101     setMethod vSet
102     If(exists(q))
103         Discard(q)
104     EndIf
105     If(vCalc="C")
106         RegionSetFocus("vMath.B2")
107     Else
108         RegionSetFocus("vMath.B1")
109     EndIf
110 Caseof "FlipBttn": // the "Flip #s" button
111     vReverse SwitchLists
112     If(exists(q))
```

```
113     Discard(q)
114 EndIf
115 RegionSetFocus("vMath.B1")
116 Caseof "PageBttn": // the "New Page" button
117     OnError Call(vError) FileNew OnError
118     If(vCalc="C")
119         RegionSetFocus("vMath.B2")
120     Else
121         RegionSetFocus("vMath.B1")
122     EndIf
123 Caseof "UseBttn1": var=1 setUse // new on 6/4/2004; inserts result in top box
124 Caseof "UseBttn2": var=2 setUse // new on 6/4/2004; inserts result in bottom box
125 EndSwitch Return
126 Label(setMethod) // this routine shows/hides/resets some items when the upper left list box is clicked
127 var=RegionGetSelectedText("vMath.B0") vMethod=SubStr(var;1;1)
128 If(vMethod<>"N") vSpecial=0 EndIf
129 Switch(vMethod) // the user's selection will be Numbers, Dates, or Lawyer dates N, D, L
130     Caseof "N":
131         If(exists(saveT2)) //new on 6/4/2004; uses stored value when switching around between groups
132             RegionSetWindowText("vMath.B2";saveT2) Discard(saveT2) EndIf
133         RegionShowWindow("vMath.S10";Hide!) RegionShowWindow("vMath.S5";Show!)
134         RegionShowWindow("vMath.B4";Show!) RegionShowWindow("vMath.FlipBttn";Show!)
135         RegionShowWindow("vMath.2006Bttn";Hide!)
136         RegionSetWindowText("vMath.S0";"1st number:")
137         RegionSetWindowText("vMath.S1";"2nd number:")
138         Nsel=RegionGetSelectedText("vMath.B3") vCalc=substr(Nsel;1;1)
139         Switch(Nsel)
140             Caseof "Cumul total":
141                 saveT2=tempn2 //new on 6/4/2004; uses stored value when switching around between groups
142                 RegionShowWindow("vMath.Ssel";Hide!)
143                 RegionSetWindowText("vMath.S5";"Round ALL NUMBERS to how many decimals?")
144                 RegionShowWindow("vMath.S3";Hide!)
145                 RegionShowWindow("vMath.UseBttn1";Hide!) // new on 6/4/2004
146                 RegionShowWindow("vMath.UseBttn2";Hide!) // new on 6/4/2004
147                 RegionShowWindow("vMath.SUse";Hide!) // new on 6/4/2004
148                 RegionShowWindow("vMath.B1";Hide!)
149                 RegionShowWindow("vMath.S11";Show!) RegionShowWindow("vMath.S0";Hide!)
150                 RegionShowWindow("vMath.FlipBttn";Hide!)
151                 RegionSetWindowText("vMath.S1";"New number:")
152                 RegionSetWindowText("vMath.B2";"") RegionSetFocus("vMath.B2")
153                 If(exists(Ttxt2))
154                     RegionSetWindowText("vMath.S6";Ttxt2)
155                 Else
156                     RegionSetWindowText("vMath.S6";"Nothing to compute - Select the decimal
157                     desired for this session and then enter a number in the top box")
158                 If(exists(vResult)) Discard(vResult) EndIf
159             EndIf
160         Default:
161             RegionShowWindow("vMath.Ssel";Show!)
162             RegionSetWindowText("vMath.S5";"Round the RESULT to how many decimals?")
163             RegionShowWindow("vMath.S3";Show!)
164             RegionShowWindow("vMath.UseBttn1";Show!) // new on 6/4/2004
165             RegionShowWindow("vMath.UseBttn2";Show!) // new on 6/4/2004
166             RegionShowWindow("vMath.SUse";Show!) // new on 6/4/2004
167             RegionShowWindow("vMath.FlipBttn";Show!)
168             RegionShowWindow("vMath.B1";Show!)
```



```

169         RegionShowWindow("vMath.S11"; Hide!) RegionShowWindow("vMath.S0"; Show!)
170         RegionSetWindowText("vMath.S0"; "1st number:")
171         RegionSetWindowText("vMath.S1"; "2nd number:") RegionSetFocus("vMath.B1")
172         RegionSetWindowText("vMath.S6"; "")
173         tempN1 = RegionGetWindowText("vMath.B1")
174         tempN2 = RegionGetWindowText("vMath.B2")
175     EndSwitch
176     If(Nsel="Ratio to total" or Nsel="Percentage of 1st")
177         RegionShowWindow("vMath.sPer"; Show!)
178     Else
179         RegionShowWindow("vMath.sPer"; Hide!)
180     EndIf
181     Caseof "D": //revised 6/4/2004
182         RegionShowWindow("vMath.sPer"; Hide!)
183         RegionShowWindow("vMath.S10"; Show!) RegionShowWindow("vMath.S5"; Hide!)
184         RegionShowWindow("vMath.2006Bttn"; Hide!)
185         RegionShowWindow("vMath.UseBttn1"; Hide!) // new on 6/4/2004
186         RegionShowWindow("vMath.UseBttn2"; Hide!) // new on 6/4/2004
187         RegionShowWindow("vMath.SUse"; Hide!) // new on 6/4/2004
188         RegionShowWindow("vMath.B1"; Show!)
189         RegionShowWindow("vMath.S11"; Hide!) RegionShowWindow("vMath.S0"; Show!)
190         RegionShowWindow("vMath.B4"; Hide!) RegionShowWindow("vMath.FlipBttn"; Hide!)
191         RegionShowWindow("vMath.2006Bttn"; Hide!)
192         RegionSetWindowText("vMath.S10"; "Calculations use the 1st date to the last value and are
193         ""true"" calculations. No adjustment is made for dates ending on weekends. For that, use
194         Lawyer dates.") RegionSetWindowText("vMath.S0"; "Date value:")
195         var = RegionGetSelectedText("vMath.B3") var = substr(var; 1; 1)
196         If(var="D" or var="W" or var="M" or var="Y")
197             RegionSetWindowText("vMath.S1"; "Date value:")
198         Else
199             RegionSetWindowText("vMath.S1"; "Second value:")
200         EndIf
201         Dsel = RegionGetSelectedText("vMath.B3") x = substr(Dsel; 1; 1)
202         tempD2 = RegionGetWindowText("vMath.B2")
203         y = strpos(tempD2; "/" )
204         Switch(x)
205             Caseof "D"; "W"; "M"; "Y":
206                 If(y>0)
207                     RegionSetWindowText("vMath.B2"; tempD2) Else
208                     RegionSetWindowText("vMath.B2"; "")
209                 EndIf
210             Default:
211                 If(y>0)
212                     RegionSetWindowText("vMath.B2"; "0") Else
213                     RegionSetWindowText("vMath.B2"; tempD2)
214                 EndIf
215             EndSwitch
216         RegionSetFocus("vMath.B2")
217     Caseof "L":
218         RegionShowWindow("vMath.sPer"; Hide!)
219         RegionShowWindow("vMath.S10"; Show!) RegionShowWindow("vMath.S5"; Hide!)
220         tempL1 = RegionGetWindowText("vMath.B1")
221         tempL2 = RegionGetWindowText("vMath.B2")
222         RegionShowWindow("vMath.2006Bttn"; Show!)
223         RegionShowWindow("vMath.UseBttn1"; Hide!) // new on 6/4/2004
224         RegionShowWindow("vMath.UseBttn2"; Hide!) // new on 6/4/2004

```

```
225     RegionShowWindow("vMath.SUse";Hide!) // new on 6/4/2004
226     RegionShowWindow("vMath.S10";Show!)
227     RegionShowWindow("vMath.B1";Show!)
228     RegionShowWindow("vMath.S11";Hide!) RegionShowWindow("vMath.S0";Show!)
229     RegionShowWindow("vMath.B4";Hide!) RegionShowWindow("vMath.FlipBttn";Hide!)
230     RegionShowWindow("vMath.2006Bttn";Show!)
231     RegionSetWindowText("vMath.S10"; "The date is day 'zero' for computing purposes. If an
232     end date falls on a weekend or legal holiday, the end date is usually the next business day.
233     12 OS §2006. Weekends are adjusted.") RegionSetWindowText("vMath.S0";"Date value:")
234     RegionSetWindowText("vMath.S1";"Number value:")
235     Lsel=RegionGetSelectedText("vMath.B3")
236 EndSwitch
237 Return
238     Label(resetList) // this routine is called when the Numbers/Dates/Lawyer dates list item is clicked and
239     completely resets control B3's list and some other stuff
240     var=RegionGetSelectedText("vMath.B0") var=Substr(var;1;1)
241     If(vMethod<>"N") vSpecial=0 EndIf
242     If(vMethod=var) // if vMethod is already the same value, no need to reset the stuff
243     Return
244 EndIf
245
246 RegionResetList("vMath.B3")
247 Switch(var)
248     Caseof "N": // this resets the list box items if "Numbers" is selected in the list box
249         RegionAddListItem("vMath.B3";"Cumul total")
250         RegionAddListItem("vMath.B3";"Add to 1st")
251         RegionAddListItem("vMath.B3";"Subtract from 1st")
252         RegionAddListItem("vMath.B3";"Multiply by 1st")
253         RegionAddListItem("vMath.B3";"Divide into 1st")
254         RegionAddListItem("vMath.B3";"Percentage of 1st")
255         RegionAddListItem("vMath.B3";"Ratio to total")
256         RegionSelectListItem("vMath.B3";Nsel)
257         Switch(Nsel)
258             Caseof "Cumul total": RegionSetWindowText("vMath.B2";"")
259                 If(exists(Ttxt2))
260                     RegionSetWindowText("vMath.S6";Ttxt2)
261                 Else
262                     RegionSetWindowText("vMath.S6";"")
263             EndIf
264             RegionSelectListItem("vMath.B3";"Cumul total")
265             RegionSetWindowText("vMath.B2";"")
266         Default:
267             If(Exists(Nsel))
268                 var=Nsel Else var= "Add to 1st"
269             EndIf
270             RegionSelectListItem("vMath.B3";Nsel)
271             RegionSetWindowText("vMath.B1";tempn1)
272             RegionSetWindowText("vMath.B2";tempn2)
273         EndSwitch
274     RegionShowWindow("vMath.S5";Show!) RegionShowWindow("vMath.B4";Show!)
275     RegionShowWindow("vMath.S5";Hide!)
276     RegionShowWindow("vMath.Ssel";Show!) RegionShowWindow("vMath.B3";Show!)
277     RegionShowWindow("vMath.B4";Show!) RegionShowWindow("vMath.S10";Show!)
278     RegionShowWindow("vMath.2006Bttn";Hide!)
279     RegionShowWindow("vMath.B1";Show!) RegionShowWindow("vMath.S11";Hide!)
280     Caseof "D": // this resets the list box items if "Dates" is selected in the list box
```

```
281 RegionShowWindow("vMath.B1"; Show!)
282 RegionShowWindow("vMath.S11"; Hide!) RegionShowWindow("vMath.S0"; Show!)
283 RegionSetWindowText("vMath.B1"; tempd1)
284 RegionSetWindowText("vMath.B2"; tempd2)
285 RegionShowWindow("vMath.B3"; Show!)
286 RegionAddListItem("vMath.B3"; "Days between")
287 RegionAddListItem("vMath.B3"; "Weeks between")
288 RegionAddListItem("vMath.B3"; "Months between")
289 RegionAddListItem("vMath.B3"; "Years between")
290 RegionAddListItem("vMath.B3"; "Add days")
291 RegionAddListItem("vMath.B3"; "Add weeks")
292 RegionAddListItem("vMath.B3"; "Add months")
293 RegionAddListItem("vMath.B3"; "Add years")
294 RegionAddListItem("vMath.B3"; "Subtract days")
295 RegionAddListItem("vMath.B3"; "Subtract weeks")
296 RegionAddListItem("vMath.B3"; "Subtract months")
297 RegionAddListItem("vMath.B3"; "Subtract years")
298 If(Exists(Dsel))
299     var=Dsel
300     else
301     var="Add days"
302 EndIf
303 RegionSelectListItem("vMath.B3"; var)
304 RegionShowWindow("vMath.S5"; Hide!) RegionShowWindow("vMath.B4"; Hide!)
305 RegionShowWindow("vMath.2006Btn"; Hide!)
306 Caseof "L": // this resets the list box items if "Lawyer dates" is selected in the list box
307 RegionShowWindow("vMath.UseBtn1"; Hide!) // new on 6/4/2004
308 RegionShowWindow("vMath.UseBtn2"; Hide!) // new on 6/4/2004
309 RegionShowWindow("vMath.SUse"; Hide!) // new on 6/4/2004
310 RegionShowWindow("vMath.B1"; Show!)
311 RegionShowWindow("vMath.S11"; Hide!)
312 RegionShowWindow("vMath.S0"; Show!)
313 RegionShowWindow("vMath.2006Btn"; Show!)
314 RegionShowWindow("vMath.B1"; Show!)
315 RegionShowWindow("vMath.S11"; Hide!) RegionShowWindow("vMath.S0"; Show!)
316 RegionSetWindowText("vMath.B1"; tempL1)
317 RegionSetWindowText("vMath.B2"; tempL2)
318 RegionShowWindow("vMath.S5"; Hide!) RegionShowWindow("vMath.B4"; Hide!)
319 RegionAddListItem("vMath.B3"; "Add days")
320 RegionAddListItem("vMath.B3"; "Add weeks")
321 RegionAddListItem("vMath.B3"; "Add months")
322 RegionAddListItem("vMath.B3"; "Add years")
323 If(Exists(Lsel))
324     var=Lsel
325     Else
326     var="Add days"
327 EndIf
328 RegionSelectListItem("vMath.B3"; var)
329 RegionShowWindow("vMath.S5"; Hide!)
330 RegionShowWindow("vMath.S3"; Show!) RegionShowWindow("vMath.Ssel"; Show!)
331 RegionShowWindow("vMath.B3"; Show!) RegionShowWindow("vMath.B4"; Hide!)
332 RegionShowWindow("vMath.S10"; Hide!) RegionShowWindow("vMath.2006Btn"; Show!)
333 EndSwitch
334 Return
335 Label(SwitchLists) // this tells the macro what to "Call" - the shorthand method for Call is used
336 Switch(vMethod)
```

```
337   Caseof "N": DoMath
338   Caseof "D": DoDates
339   Caseof "L": DoLawyer
340 EndSwitch
341 Return
342   Label(setUse) // new routine on 6/4/2004: inserts the result into the top or bottom math box, if a result exists
343 If(exists(vResult))
344   x=StrPos(vResult,"%")
345   If(x>0)
346     tempn=StrTransform(vResult,"%";"") // new on 6/4/2004: this removes the % symbol, if present
347     tempn=StrTransform(tempn;",";"") // new on 6/4/2004: if ",", symbol is present, removes commas
348     tempn=.01*tempn // new on 6/4/2004: this converts % values to regular numeric values
349   EndIf
350   Switch(var)
351     Caseof 1:
352       If(x=0)
353         RegionSetWindowText("vMath.B1";vResult) tempn1=vResult
354       Else
355         RegionSetWindowText("vMath.B1";tempn) tempn1=tempn
356       EndIf
357     Caseof 2:
358       If(x=0)
359         RegionSetWindowText("vMath.B2";vResult) tempn2=vResult
360       Else
361         RegionSetWindowText("vMath.B2";tempn) tempn2=tempn
362       EndIf
363   EndSwitch
364   Else DialogShow("NoResult";"vMath") DialogDestroy("NoResult")
365 EndIf
366 Return
367   Label(setResult)
368 If(exists(q))
369   RegionSetWindowText("vMath.G1";"Error / Info Message")
370 Else
371   If(vSpecial>1) // revised 6/4/2004 to deal with scientific notation results or digits >= 15
372     RegionSetWindowText("vMath.G1";"APPROXIMATE Result * * * Click Help for more detail")
373   Else
374     RegionSetWindowText("vMath.G1";"Result")
375   EndIf
376 EndIf
377 Return
378   Label(doMath) // the called routines here are for "numbers", not the "dates" or "lawyer dates"
379 vFinal=0 e=0
380 If(NSel<>"Cumul total")
381   num1=RegionGetWindowText("vMath.B1")
382 Else num1=saveResult
383 EndIf
384 num2=RegionGetWindowText("vMath.B2")
385 varD=RegionGetSelectedText("vMath.B4") varD=StrNum(varD)
386 vCalc=RegionGetSelectedText("vMath.B3") Nsel=vCalc vCalc=Substr(vCalc;1;1)
387 If(num1="" and num2="")
388   q=1
389   RegionSetWindowText("vMath.S6";"Nothing to compute - enter numbers in any needed
390   boxes")
391   If(vCalc="C")
392     RegionSetFocus("vMath.B2")
```

```
393     else
394         RegionSetFocus("vMath.B1")
395     EndIf
396     setResult Return
397 EndIf
398 If(num1="")
399     num1="0"
400 EndIf
401 var=num1 Call(chkNum)
402 If(exists(q))
403     RegionSetWindowText("vMath.S6";"1st number, "+var+", is not a number. Fix it.")
404     RegionSetFocus("vMath.B1")
405     RegionSetEditSelection("vMath.B1") setResult Return
406 EndIf
407 var=num1
408 var=StrTransform(var;";";"") // replacement routine for getting rid of commas
409 chkMax // error routine new 6/4/2004
410 If(exists(q))
411     RegionSetWindowText("vMath.S6";"1st number, "+var+", is too long. Grande Math cannot
412     work with numbers exceeding 15 digits - unreliable results may occur. Use a different
413     number.") RegionSetFocus("vMath.B1") RegionSetEditSelection("vMath.B1") Return
414 EndIf
415 num1=var // num1 is now a legal raw number, w/o commas
416 Call(NewNum) //var will now have commas; num1 is original var, without commas
417 If(Nsel<>"Cumul total")
418     FirstN=var tempn1=FirstN //FirstN is declared, with commas; FirstN is used when typing
419 EndIf
420 If(num2="")
421     num2="0"
422 EndIf
423 var=num2 Call(chkNum)
424 If(exists(q))
425     RegionSetWindowText("vMath.S6";"2nd number, "+var+", is not a number. Fix it.")
426     RegionSetFocus("vMath.B2")
427     RegionSetEditSelection("vMath.B2") setResult
428     Return
429 EndIf
430 var=num2
431 var=StrTransform(var;";";"") // replaces previous routine to get rid of commas
432 chkMax // error routine new 6/4/2004
433 If(exists(q))
434     RegionSetWindowText("vMath.S6";"2nd number, "+var+", is too long. Grande Math cannot
435     work with numbers exceeding 15 digits - unreliable results may occur. Use a different
436     number.") RegionSetFocus("vMath.B2") RegionSetEditSelection("vMath.B2")
437     Return
438 EndIf
439 num2=var
440 Call(NewNum) //new in 6/4/2004; corrects display format
441 If(Nsel<>"Cumul total")
442     SecondN=var tempn2=var
443 EndIf
444 num2=StrNum(num2)
445 UsePer=0 // this variable declares the default intention of not using percentage signs when typing in doc
446 Switch(vCalc)
447     Caseof "A": var=num1+num2 // simple addition
448     Caseof "C": // this does the cumulative total using workarounds for floating cell errors
```

```
449     If(exists(T[]))
450         Else Declare(T[1000])
451     EndIf
452     If(varD=0)
453         num2= Integer(roundoff(num2; 1/(10**varD))) Else
454         num2=numstr(num2) x=strPos(num2; ".") If(x>0) num2=num2+"000001"
455     EndIf
456     num2=strnum(num2)
457     num2 = RoundOff (num2; (1/(10**varD))) EndIf
458     var=num2 Call(newNum) SecondN=var
459     If(exists(yResult))
460         var=num2+yresult yresult=var
461     Else var=num2 yResult=num2
462     EndIf
463     RealTotal=var
464     Caseof "S": var=num1-num2 // this does simple subtraction
465     Caseof "M": var=num1*num2 // this is simple multiplication
466     Caseof "D": // this is ordinary division
467         If(num2=0)
468             q=1
469             RegionSetWindowText("vMath.S6"; "2nd number is 0. Division by 0 cannot be done.")
470             RegionSetFocus("vMath.B2") setResult
471             Return
472         EndIf
473         var=num1/num2
474     Caseof "P": // this is division, but, with the eventually called Label (not here), will reset values to % values
475         If(num1=0)
476             q=1
477             RegionSetWindowText("vMath.S6"; "1st number is 0. Division by 0 cannot be done.")
478             RegionSetFocus("vMath.B1") setResult
479             Return
480         EndIf
481         var=num2/num1 UsePer=1
482     Caseof "R": // essentially the same as the above, but uses different values for computation
483         If(num1+num2=0)
484             q=1
485             RegionSetWindowText("vMath.S6"; "The total is 0. Division by 0 cannot be done.")
486             RegionSetFocus("vMath.B1") setResult
487             Return
488         EndIf
489         var=num2/(num1+num2) UsePer=1
490     EndSwitch
491     vSpecial=0 // added routines 6/4/2004 to deal with scientific notation and other >=15 digit issues
492     x=strpos(var; "-")
493     If(x>0)
494         var=var*-1 negnum=1
495     else negnum=0
496     EndIf
497     If(var<1) // for results with values less than 1
498         vSpecial=1
499     EndIf
500     x=strpos(var; "e") If(x>0) e=1 vSpecial=2 EndIf // for results that are in scientific notation
501
502     x=var
503     If(vSpecial=0)
504         tVar=numstr(x; 0) xvar=numstr(x)
```



```

505   If(0+tvar=xVar) var=tVar EndIf // eliminates any decimal and trailing zeros & var becomes string
506   x=strpos(var;".")
507   If(x>0)
508       vMainL=substr(var;1;x-1) vMainL=strlen(vMainL)
509       vRemL=substr(var;x+1;strlen(var)) vRemL=strlen(vRemL) vAll=vMainL+vAll
510   Else
511       vRemL=0
512       vMainL=strlen(var) vAll=vMainL vRemL=0
513   EndIf
514   num1F=FractionalPart(num1) num2F=FractionalPart(num2)
515   vAll=StrTransform(var;".","") vAll=strlen(vAll)
516   rMax=15-(vAll) // rMax is the maximum rounding capacity
517   If(Num1F>0 or Num2F>0) NumF=1 Else NumF=0 EndIf // testing for either's fractional value
518 EndIf
519 //... Identifying the 3's after this point ... otherwise, all others will be vSpeicals will be 0
520 If(vAll=15 and NumF=1)
521     If(varD>vRemL)
522         vSpecial=3 e=1
523     EndIf
524 EndIf
525 If(vSpecial=0)
526     If(varD>rMax)
527         If(vRemL<varD)
528             If(vAll<15)
529                 Else
530                     If(vRemL<>0)
531                         vSpecial=3
532                     EndIf
533                 EndIf
534             EndIf
535         EndIf
536     EndIf
537 EndIf
538 // new & revised 6/4/2004: replaces earlier version's RoundOff method: deals w/ floating cell & scientific notation issues
539 Switch(vSpecial)
540     Caseof 0:
541         If(negnum=1) // modified 6/4/2004: negnum set above
542             RealTotal="-"+var // RealTotal added 6/4/2004
543         Else
544             RealTotal=var
545         EndIf
546     Switch(varD)
547         Caseof 0: If(usePer=0) var=RoundOff(var;1) else var=RoundOff(var;0.01) EndIf
548         Caseof 1:
549             If(usePer=0) var=RoundOff(var;0.01) else var=RoundOff(var;0.001) EndIf
550             var=RoundOff(var;0.1)
551         Default:
552             vPoints=varD
553             x="0."
554             z=1
555             Repeat
556                 x=x+"0"
557                 z=z+1
558             Until (z=vPoints)
559             x=x+"1"
560             x=strnum(x)

```

```
561         var=RoundOff(var;x)
562     EndSwitch
563     If(negnum=1)
564         var="-"+var
565     EndIf
566     Caseof 1: // for results less than 1, greater than 0: negnum was set above
567         If(negnum=1)
568             RealTotal="-"+var // RealTotal added 6/4/2004
569         Else
570             RealTotal=var
571         EndIf
572         var=numstr(var;varD) var=strnum(var)
573     Caseof 2: // for results containing scientific notation
574         x=strpos(var;"e")
575         y=substr(var;x+2;3) y=strnum(y) y=y+1
576         var=substr(var;1;x-1)
577         var=var-". "
578         varq=strlen(var)
579         If(varq<y)
580             Repeat
581                 var=var+"0"
582             Until (strlen(var)=y)
583         EndIf
584         x=strpos(var;".")
585         If(x=0)
586             var=var-". "
587         EndIf
588         If(negnum=1)
589             RealTotal="-"+var // RealTotal added 6/4/2004
590         Else
591             RealTotal=var
592         EndIf
593     Caseof 3: // for results greater than 1: negnum was set above
594         If(negnum=1) RealTotal="-"+var
595         Else
596             RealTotal=var
597         EndIf
598 EndSwitch
599 If(vSpecial<>2)
600     var=strnum(var)
601     tVar=numstr(var;0) xvar=numstr(var)
602     If(0+tvar=xVar)
603         var=tVar
604     EndIf // added 6/4/2004: eliminates trailing zeros in displayed result for non-scientific notation results
605 EndIf
606 NewNum // this converts the result into a string
607 If(UsePer=1)
608     Call(MakePer)
609 EndIf
610 vResult=var
611 If(vSpecial=2 or vSpecial=3)
612     x=0
613     negnum=strpos(var;"-")
614     If(negnum>0)
615         x=x+1
616     EndIf
```

```
617   y=strpos(var; ".")
618   If(y>0)
619     x=x+1
620   EndIf
621   v15=substr(RealTotal; 1; 15+x)
622 EndIf
623 If(vCalc="C")
624   If(exists(vN))
625     vN=vN+1 Else vN=1
626   EndIf
627   T[vN] = SecondN
628   saveResult=vResult
629   RegionSetWindowText("vMath.B2"; "")
630   RegionSetFocus("vMath.B2")
631   If(exists(Ttxt))
632     Ttxt=Ttxt+ " + " +SecondN Else
633     Ttxt=vResult
634   EndIf
635   If(Ttxt=vResult)
636     Ttxt2=vResult Else
637     Ttxt2=Ttxt + " = " +vresult
638   EndIf
639   RegionSetWindowText("vMath.S6"; Ttxt2)
640   Else
641     RegionSetWindowText("vMath.B1"; FirstN)
642     RegionSetWindowText("vMath.B2"; SecondN)
643     RegionSetWindowText("vMath.S6"; vResult)
644   EndIf
645   setResult //resets info box top line
646   Return
647   Label(doDates) // this routine is called for "Dates", not "Numbers" or "Lawyer dates"
648   var=RegionGetWindowText("vMath.B1") tempD1=var
649   var2=RegionGetWindowText("vMath.B2") tempD2=var2
650   If(var="")
651     q=1
652     RegionSetWindowText("vMath.S6"; "Nothing to compute - enter date in top box")
653     RegionSetEditSelection("vMath.B1") setResult Return
654   EndIf
655   x=substr(dsel; 1; 1)
656   Switch(x)
657     Caseof "D"; "W"; "Y"; "M":
658       If(tempD2="")
659         RegionSetWindowText("vMath.S6"; "Nothing to compute - enter date in second box")
660         RegionSetEditSelection("vMath.B2") setResult Return
661       EndIf
662     Default:
663   EndSwitch
664   Call(dateCnv)
665   If(Exists(q))
666     RegionSetWindowText("vMath.S6"; vMsg)
667     RegionSetFocus("vMath.B2")
668     RegionSetEditSelection("vMath.B2") setResult Return
669   EndIf
670   FirstN=var tempd1=mo+ "/" + day+ "/" + year
671   var=RegionGetSelectedText("vMath.B3") Dsel=var vCalc=substr(dSel; 1; 1)
672   Switch(var)
```

```
673 Caseof "Days between": vCalc=1 getDays
674     If(Exists(q))
675         Return
676     EndIf
677     vBetween
678     var=days Call(NewNum) vDays=var vResult=vDays+ " days"
679 Caseof "Weeks between": vCalc=2 getDays
680     If(Exists(q))
681         Return
682     EndIf
683     vBetween
684     Repeat
685         saveback=t
686         w=DateAddWeeks(saveback;0)
687         x=DateAddWeeks(t;1)
688         y=Integer(ConvertType(x;Float!)) t=y z=z+1
689     Until (y>=s)
690     If(y>s) // If so, we want to go back 1 mo and count days from that to end of 2nd date
691         z=z-1 // and this subtracts 1 week from total years if y's value exceeds s's value
692         x=Integer(ConvertType(saveback;Float!)) vDays=s-x
693     Else vDays=0
694     EndIf
695     If(vDays=1)
696         dword="day" Else
697         dword="days"
698     EndIf
699     var=z Call(NewNum) vWeeks=var
700     If(vWeeks=1)
701         wword="week" else wword="weeks"
702     EndIf
703     vResult=vWeeks+ " "+wword+ ", "+vDays+ " "+dword
704 Caseof "Months between": vCalc=3 getDays
705     If(Exists(q))
706         Return
707     EndIf
708     Call(vBetween)
709     Repeat
710         saveback=t
711         w=DateAddMonths(saveback;0)
712         x=DateAddMonths(t;1)
713         y=Integer(ConvertType(x;Float!)) t=y z=z+1
714     Until (y>=s)
715     vMonths=z
716     If(y>s) // If so, we want to go back 1 mo and count days from that to end of 2nd date
717         vMonths=z-1 // and this subtracts 1 month from total months in y's value
718         x=Integer(ConvertType(saveback;Float!)) vDays=s-x
719     Else vDays=0
720     EndIf
721     If(vDays=1)
722         dword="day" else dword="days"
723     EndIf
724     If(vMonth=1)
725         mword="month" else mword="months"
726     EndIf
727     var=vMonths Call(NewNum) vMonths=var
728     vResult=vMonths+ " "+mword+ ", "+vDays+ " "+dword
```

```
729 Caseof "Years between": vCalc=4 getDays
730     If(Exists(q))
731         Return
732     EndIf
733     If(d=s)
734         vYears=0 vMonths=0 vDays=0 Else
735         Call(vBetween)
736         Repeat
737             saveback=t
738             w=DateAddYears(saveback;0)
739             x=DateAddYears(t;1)
740             y=Integer(ConvertType(x;Float!)) t=y z=z+1
741         Until (y>=s)
742         vYears=z
743         If(y=s)
744             vMonths=0 vDays=0
745             Else
746                 z=0
747                 vYears=vYears-1
748                 t=Integer(ConvertType(saveback;Float!))
749                 Repeat
750                     saveback=t
751                     w=DateAddMonths(saveback;0)
752                     x=DateAddMonths(t;1)
753                     y=Integer(ConvertType(x;Float!)) t=y z=z+1
754                 Until (y>s)
755                 If(y>s)
756                     vMonths=z-1
757                     x=Integer(ConvertType(saveback;Float!))
758                     vDays=s-x
759                     Else vDays=0
760                 EndIf
761             EndIf
762         EndIf
763         If(vYears=1)
764             yword="year" else yword="years"
765         EndIf
766         If(vMonths=1)
767             mword="month" else mword="months"
768         EndIf
769         If(vDays=1)
770             dword="day" else dword="days"
771         EndIf
772         vResult=vYears+" "+yword+", "+vMonths+" "+mword+", "+vDays+" "+dword
773 Caseof "Add days": vCalc=5 getV
774     If(Exists(q))
775         Return
776     EndIf
777     d=DateAddDays(DateAndTime (Day;Mo;Year;0;0;0;0); var) set6
778 Caseof "Add weeks": vCalc=6 getV
779     If(Exists(q))
780         Return
781     EndIf
782     d=DateAddWeeks(DateAndTime (Day;Mo;Year;0;0;0;0);var) set6
783 Caseof "Add months": vCalc=7 getV
784     If(Exists(q))
```

```
785     Return
786     EndIf
787     d=DateAddMonths(DateAndTime (Day;Mo;Year;0;0;0;0); var) set6
788     Caseof "Add years": vCalc=8 getV
789     If(exists(q))
790         Return
791     EndIf
792     d=DateAddYears(DateAndTime (Day;Mo;Year;0;0;0;0); var) set6
793     Caseof "Subtract days": vCalc=9 getV
794     If(exists(q))
795         Return
796     EndIf
797     d=DateAddDays(DateAndTime (Day;Mo;Year;0;0;0;0); -var) set6
798     Caseof "Subtract weeks": vCalc=10 getV
799     If(exists(q))
800         Return
801     EndIf
802     d=DateAddWeeks(DateAndTime (Day;Mo;Year;0;0;0;0); -var) set6
803     Caseof "Subtract months": vCalc=11 getV
804     If(exists(q))
805         Return
806     EndIf
807     // DateAddMonths is broken in Wp9, 10 and 11 insofar as subtracting months if the year would be earlier than the year
808     containing the date value; this work-around code draws from suggestions by Ken Hobson in a WordPerfect Universe
809     thread on 9/17/2002: use the following structure when subtracting months, where "var" is the number of months
810     x=DateAndTime(Day;Mo;Year;0;0;0;0)
811     yMonths= var Mod 12 yYears= (var-yMonths) /12
812     d=DateAddYears(DateAddMonths(x; 12-yMonths); -1-yYears) set6
813     Caseof "Subtract years": vCalc=12 getV
814     If(exists(q))
815         Return
816     EndIf
817     d=DateAddYears(DateAndTime (Day;Mo;Year;0;0;0;0); -var) set6
818 EndSwitch
819 RegionSetWindowText("vMath.S6";vResult) setResult
820 Return
821 Label(doLawyer) // this routine is called for "Lawyer dates", not "Numbers" or "Dates"
822 var=RegionGetWindowText("vMath.B1") templ1=var
823 var2=RegionGetWindowText("vMath.B2") templ2=var2
824 If(var="")
825     var="0"
826 EndIf
827 If(var2="")
828     var2="0" RegionSetFocus("vMath.B2")
829 EndIf
830 If(var="0")
831     q=1
832     RegionSetWindowText("vMath.S6";"Nothing to compute - enter values in any needed boxes")
833     RegionSetFocus("vMath.B1") RegionSetEditSelection("vMath.B1") setResult Return
834 EndIf
835 Call(dateCnv)
836 If(Exists(q))
837     RegionSetWindowText("vMath.S6";vMsg) RegionSetFocus("vMath.B1")
838     RegionSetEditSelection("vMath.B1")
839     setResult Return
840 EndIf
```

```
841 FirstN=var tempL1=mo+ "/" +day+ "/" +year
842 var=RegionGetSelectedText("vMath.B3") Lsel=var
843 Switch(var)
844   Caseof "Add days": vCalc=1 getV
845     If(exists(q))
846       Return
847     EndIf
848     d=DateAddDays(DateAndTime (Day;Mo;Year;0;0;0;0); var) set6
849   Caseof "Add weeks": vCalc=2 getV
850     If(exists(q))
851       Return
852     EndIf
853     d=DateAddWeeks(DateAndTime (Day;Mo;Year;0;0;0;0); var) set6
854   Caseof "Add months": vCalc=3 getV
855     If(exists(q))
856       Return
857     EndIf
858     d=DateAddMonths(DateAndTime (Day;Mo;Year;0;0;0;0); var) set6
859   Caseof "Add years": vCalc=4 getV
860     If(exists(q))
861       Return
862     EndIf
863     d=DateAddYears(DateAndTime (Day;Mo;Year;0;0;0;0); var) set6
864   EndSwitch
865 RegionSetWindowText("vMath.S6";vResult) setResult
866 Return
867   Label(vBetween) // this called routine converts m/d/yyyy dates into serial date values
868   d=DateAndTime (day1; mo1;year1;0;0;0;0) d=Integer(ConvertType(d;Float!))
869   s=DateAndTime (day2;mo2;year2;0;0;0;0) s=Integer(ConvertType(s;Float!))
870   If(d>s)
871     x=d y=s s=x d=y day1=day2 mo1=mo2 year1=year2
872   EndIf
873   days=Integer(s-d)
874   t=d z=0 // z is number of months and t becomes d, and d remains unchanged
875   Return
876   Label(getDays) // this routine is called from Label(doDates); sets text appearing in the Results box, Region.S6
877   mo1=mo day1=day year1=year
878   var=RegionGetWindowText("vMath.B2")
879   If(var="")
880     q=1
881     RegionSetWindowText("vMath.S6";"Nothing to compute - enter values in any needed boxes")
882     RegionSetFocus ("vMath.B1") RegionSetEditSelection("vMath.B1") setResult Return
883   EndIf
884   Call(dateCnv) SecondN=var
885   If(exists(q))
886     RegionSetWindowText("vMath.S6";vMsg) RegionSetFocus("vMath.B2")
887     RegionSetEditSelection("vMath.B2") Return
888   EndIf
889   day2=day mo2=mo year2=year
890   Return
891   Label(getV) // this is part of the date routines and can set text in the Results box, Region.S6
892   var=RegionGetWindowText("vMath.B2")
893   If(var="")
894     var="0"
895   EndIf
896   chkNum
```



```
897 If(exists(q))
898   RegionSetWindowText("vMath.S6";"2nd value, "+var+", is not legal")
899   RegionSetFocus ("vMath.B2") RegionSetEditSelection("vMath.B2") Return
900 EndIf
901 If(NegNum=1)
902   q=1
903   RegionSetWindowText("vMath.S6";"2nd value, "+var+", is not legal. Use Subtract in regular
904   date list instead.")
905   Discard(NegNum)
906   RegionSetFocus ("vMath.B2") RegionSetEditSelection("vMath.B2") setResult Return
907 EndIf
908 var=StrNum(var)
909 SecondN=tempD2
910 Return
911   Label(set6) // this is part of the lawyer date routines in case the date falls on a weekend
912 If(vMethod="L")
913   x=DateWeekdayName(d;Short!)
914   If (x="Sat")
915     d=d+2
916   EndIf
917   If(x="Sun")
918     d=d+1
919   EndIf
920 EndIf
921 d=DateString(d; "dddd, MMMM d, yyyy") vResult=d Return
922   Label(vReverse) // this flip-flops the values in numbers 1 and 2
923 Num1=RegionGetWindowText("vMath.B1") Num2=RegionGetWindowText("vMath.B2")
924 RegionSetWindowText("vMath.B1";Num2) RegionSetWindowText("vMath.B2";Num1)
925 Discard(num1;num2) Return
926   Label(vSet) // this routine clears values in numbers 1, 2 and the results box
927 RegionSetWindowText("vMath.B1";"") RegionSetWindowText("vMath.B2";"")
928 If(exists(T[]))
929   While (exists(t[]))
930     Discard(t[])
931   EndWhile
932 EndIf
933 If(exists(vN))
934   Discard(vN)
935 EndIf
936 If(exists(vResult))
937   Discard(vResult)
938 EndIf
939 If(exists(yResult))
940   Discard(yResult)
941 EndIf
942 If(exists(Ttxt))
943   Discard(Ttxt)
944 EndIf
945 If(exists(Ttxt2))
946   Discard(Ttxt2)
947 EndIf
948 RegionSetWindowText("vMath.S6";"Nothing to compute - enter values in any needed boxes")
949 RegionSetFocus("vMath.B1") RegionSetEditSelection("vMath.B1") setResult Return
950   Label(vWrite) // this Label is called from Label(vType), above: it insures that decimal values are handled well
951 w=strPos(var;".")
952 Switch(w)
```

```

953 Caseof 0:
954     If(varD<>0)
955         var=var+ "." y=0
956         Repeat
957             var=var+"0" y=y+1
958         Until(y=varD)
959     EndIf
960 Default: y=substr(var;w+1;strlen(var)) z=strlen(y)
961     If(z<varD)
962         Repeat
963             var=var+"0" z=z+1
964         Until(z=varD)
965     EndIf
966 EndSwitch
967 Return
968 Label(eText) // new 6/4/2004: for results which involve scientific notation
969 Type("(after the 15th digit, the result is approximate due to computer chip limitations)") Return
970 Label(TypeM1) //new on 6/4/2004 merely to avoid typing redundancy
971 Type(t1+" "+mchar+" "+t2+" = "+vResult+" [rounded to "+varD+" "+dWord+"]") Return
972 Label(TypeDw) //new on 6/4/2004 merely to avoid typing redundancy
973 Type(Dw+" between "+SecondN+" and "+FirstN+" = "+vResult) Return
974 Label(TypeDa) //new on 6/4/2004 merely to avoid typing redundancy
975 Type(FirstN+Do+SecondN+" "+Dw+plural+" = "+vResult) Return
976 Label(vTypeE) //new on 6/4/2004: types text depending on var e
977 If(e=1)
978     Type("(actual computed result up to 15 digits)")
979 Else
980     Type("(actual computed result)")
981 EndIf
982 HardReturn
983 Return
984 Label(vTypeD) //new on 6/4/2004: part of number column typing
985 Type("= ") dTabs Type(vResult) Tab Type("(rounded to "+varD+" "+dword+")") Return
986 Label(vType) // this routine types stuff into the open document! modified to add alternative on 6/4/2004
987 RegionResetList("vType.B1")
988 Switch(vCalc)
989     Caseof "C":
990         RegionAddListItem("vType.B1";"Result only")
991         RegionAddListItem("vType.B1";"Formula and result in a column")
992         If(exists(TypeWhat))
993             Switch(TypeWhat)
994                 Caseof "All the above and sample text"; "Formula and result";"Formula and result on
995                     same line":
996                     TypeWhat="Formula and result in a column"
997             EndSwitch
998         EndIf
999     Caseof "A";"S";"M";"D";"P";"R":
1000         RegionAddListItem("vType.B1";"Result only")
1001         RegionAddListItem("vType.B1";"Formula and result in a column")
1002         RegionAddListItem("vType.B1";"Formula and result on same line")
1003         If(exists(TypeWhat))
1004             Switch(TypeWhat)
1005                 Caseof "All the above and sample text"; "Formula and result":
1006                     TypeWhat="Formula and result in a column"
1007             EndSwitch
1008         EndIf

```

```
1009 Default:
1010     RegionAddListItem("vType.B1"; "Result only")
1011     RegionAddListItem("vType.B1"; "Formula and result on same line")
1012     If(exists(TypeWhat) and (TypeWhat="Result only" or TypeWhat="Formula and result on
1013     same line")) Else
1014         TypeWhat="Result only"
1015     EndIf
1016 EndSwitch
1017 If(exists(TypeWhat))
1018     RegionSelectListItem("vType.B1"; TypeWhat) Else
1019     RegionSelectListItem("vType.B1"; "Result only")
1020 EndIf
1021 DialogShow("vType"; "WordPerfect") x=MacroDialogResult
1022 If(x=2)
1023     Return
1024 EndIf
1025 TypeMe=Strlen(TypeWhat) //substantially modified to add additional typing option on 6/4/2004
1026 x=?DocBlank
1027 If(x=false)
1028     PosLineEnd HardReturn
1029 EndIf
1030 If(vMethod="N" and varD=1)
1031     dWord="decimal"
1032 Else
1033     dWord="decimals"
1034 EndIf
1035 If(TypeMe=11) // substantially modified: result only: modified on 6/4/2004
1036     Switch(vMethod)
1037     Caseof "N":
1038         If(vCalc="C")
1039             var=yresult
1040             Type(var) RegionSetFocus("vMath.B2")
1041             Else Type(vResult) RegionSetFocus("vMath.B1")
1042         EndIf
1043         Default: Type(vResult)
1044     EndSwitch
1045     If(vSpecial>1)
1046         eText
1047     EndIf
1048 EndIf
1049 If(TypeMe=31) // substantially modified: formula & result on same line: modified 6/4/2004
1050     Switch(vMethod)
1051     Caseof "N": // what to type if numbers are being computed
1052         Switch (vCalc) // routine modified 6/4/2004 to avoid typing redundancy
1053         Caseof "A": mChar="+" t1=FirstN t2=SecondN TypeM1
1054         Caseof "S": mChar="-" t1=FirstN t2=SecondN TypeM1
1055         Caseof "M": mChar="x" t1=FirstN t2=SecondN TypeM1
1056         Caseof "D": mChar="÷" t1=FirstN t2=SecondN TypeM1
1057         Caseof "P": mChar="÷" t1=SecondN t2=FirstN TypeM1
1058         Caseof "R": Type(SecondN+" ÷ (" +FirstN+" + " +SecondN+" ) = "+vResult+"
1059         [rounded to "+varD+" "+dWord+""])
1060     EndSwitch
1061     Caseof "D": // what to type if regular dates are being calculated
1062         If(SecondN=1)
1063             plural="" Else plural="s"
1064         EndIf
```

```

1065 Switch(vCalc) // routine modified 6/4/2004 to avoid typing reduncancy
1066     Caseof 1: Dw="Days" TypeDw
1067     Caseof 2: Dw="Weeks" TypeDw
1068     Caseof 3: Dw="Months" TypeDw
1069     Caseof 4: Dw="Years" TypeDw
1070     Caseof 5: Dw="day" Do=" " + " TypeDa
1071     Caseof 6: Dw="week" Do=" " + " TypeDa
1072     Caseof 7: Dw="month" Do=" " + " TypeDa
1073     Caseof 8: Dw="year" Do=" " + " TypeDa
1074     Caseof 9: Dw="day" Do=" " TypeDa
1075     Caseof 10: Dw="week" Do=" " TypeDa
1076     Caseof 11: Dw="month" Do=" " TypeDa
1077     Caseof 12: Dw="year" Do=" " TypeDa
1078 EndSwitch
1079 Caseof "L": // what to type if Lawyer dates are being calculated
1080     If(SecondN=1)
1081         plural="" Else plural="s"
1082     EndIf
1083     Switch(vCalc) // routine modified 6/4/2004 to avoid typing reduncancy
1084         Caseof 1: Dw="day" Do=" " + " TypeDa
1085         Caseof 2: Dw="week" Do=" " + " TypeDa
1086         Caseof 3: Dw="month" Do=" " + " TypeDa
1087         Caseof 4: Dw="year" Do=" " + " TypeDa
1088     EndSwitch
1089 EndSwitch
1090 If(vSpecial>1)
1091     eText
1092 EndIf
1093 EndIf
1094 If(TypeMe=30) // column
1095     PosDocBottom
1096     sTabs
1097     Switch(vCalc)
1098         Caseof "C": n=1
1099         Repeat
1100             var=T[n] Call(vWrite) typevar=var
1101             dTabs Type(typevar) HardReturn n=n+1
1102         Until (T[n]="")
1103         var=vResult vWrite
1104         PosLineUp PosLineBeg vUnd Type("+") SelectMode(On!) PosLineEnd
1105         AttributeAppearanceToggle(Underline!) SelectMode(Off!) PosLineDown
1106         Type("=") dTabs Type(var) HardReturn
1107     Default: //substantially modified: formula & result in a column: added 6/4/2004
1108         Switch(vCalc)
1109             Caseof "A": vchar="+" x="Addition (rounded to "+varD+" "+dword+"): "
1110             Caseof "S": vchar="-" x="Subtraction (rounded to "+varD+" "+dword+): "
1111             Caseof "M": vchar="x" x="Multiplication (rounded to "+varD+" "+dword+" "+): "
1112             Default: vchar="÷" x="Division (rounded to "+varD+" "+dword+): "
1113         EndSwitch
1114         Switch(vCalc)
1115             Caseof "R":
1116                 var=num1+num2 newnum tResult=var
1117                 Type("Ratio of the following number to the Total of 2 Numbers (rounded to
1118                 "+varD+" "+dword+):") HardReturn
1119                 vUnd dTabs var=SecondN vWrite Type (var) PosLineEnd HardReturn
1120                 Type(vchar) HardReturn

```

```

1121     Type("[")dTabs var=FirstN vWrite Type(var) HardReturn
1122     vUnd // in Wp9 & higher, could add UnderLineStyle (Solid!) but that won't work in Wp8
1123     Type("+") dTabs var=SecondN vwrite Type(var) PosLineEnd HardReturn
1124     vUnd Type("=") dTabs var=tResult vWrite Type(var) PosLineEnd
1125     Type("]") HardReturn
1126     Type("=") dTabs var=RealTotal vWrite Type(var) PosLineEnd Tab
1127     If(e=1)
1128         Type("(actual computed result up to 15 digits)")
1129     Else
1130         Type("(actual computed result)")
1131     EndIf
1132     HardReturn
1133     Type("=") dTabs Type(vResult) Tab Type("(rounded to "+varD+" "+dword+"")")
1134     Caseof "P":
1135         Type("2nd number's percentage of 1st number (rounded to "+varD+"
1136         "+dword+"):") HardReturn
1137         dTabs Type(SecondN) HardReturn
1138         vUnd // in Wp9 & higher, could add UnderLineStyle (Solid!) but that won't work in Wp8
1139         Type(vChar) dTabs Type(FirstN) PosLineEnd HardReturn
1140         Type("=") dTabs var=RealTotal vWrite Type(var) PosLineEnd Tab
1141         vTypeE vTypeD
1142     Default:
1143         Type(x) HardReturn
1144         dTabs var=FirstN vWrite Type(var) HardReturn
1145         vUnd // in Wp9 & higher, could add UnderLineStyle (Solid!) but that won't work in Wp8
1146         Type(vchar) dTabs var=SecondN vWrite Type(var) PosLineEnd HardReturn
1147         Type("=") dTabs var=RealTotal vWrite Type(var) PosLineEnd Tab
1148         vTypeE vTypeD
1149     EndSwitch
1150     If(vSpecial>1)
1151         HardReturn eText PosLineBeg DeleteCharNext PosLineEnd
1152     EndIf
1153     EndSwitch
1154     HardReturn
1155 EndIf
1156 RegionSetFocus("vMath.B2")
1157 Return
1158     Label(dTabs) // new on 6/4/2004 to set up tabs for typing a math column
1159     x=Strlen(vResult)
1160     If(x<27) x=4 EndIf
1161     If(x>=27 and x<34) x=5 EndIf
1162     If(x>=34) x=6 EndIf
1163     RepeatValue(x) TabDecimal Return
1164     Label(sTabs)
1165     TabSet (Relative!; { 1.0"; TabLeft!; 1.5"; TabLeft!; 2.0"; TabLeft!; 2.6"; TabLeft!; 3.0"; TabLeft!;
1166     3.5"; TabLeft!; 4.0"; TabLeft!; 4.5"; TabLeft!; 5.0"; TabLeft!; 5.5"; TabLeft!; 6.0"; TabLeft!;
1167     6.5"; TabLeft!; 7.0"; TabLeft!; 7.5"; TabLeft!; 8.0"; TabLeft!})
1168     DecimalAlignmentCharacter (".") Return
1169     Label(vUnd) // new on 6/4/2004 to set up underlining when typing a math column
1170     UnderlineTabs (Yes!) UnderlineSpaces (Yes!) AttributeAppearanceToggle (Underline!) Return
1171     Label(chkMax)
1172     //new error routine 6/4/2004: the number of digits (excluding leading zeros) cannot be longer than 15 "digits"
1173     x=strtransform (var;"-";"")
1174     x=strtransform (x;".";"")
1175     x=strtrim(x;;TrimLeft!;"0")
1176     x=strlen(x)

```

```
1177 If(x>15)
1178     q=1 DialogShow("vMax"; "WordPerfect") DialogDestroy("vMax")
1179 EndIf
1180 Return
1181 Label(chkNum) // this routine determines if all characters are valid for numeric purposes. revised 6/4/2004
1182 x=StrToChars(var; Keep!; ".") x=StrLen(x) //this routine insures that only one decimal is in the string
1183 If(x>1)
1184     q=1
1185     Return
1186 EndIf
1187 x=substr(var; 1; 1)
1188 If(x= "-")
1189     NegNum=1 var=substr(var; 2; strlen(var)) Else NegNum=0
1190 EndIf
1191 Ret= IsNumberString(var)
1192 If(Ret)
1193     Else q=1
1194 EndIf
1195 Return // if the variable contains non-allowed characters, q=1 says error is present
1196
1197 Function IsNumberString (InString)
1198     RefString="1234567890., "
1199     ForNext (Count; 1; StrLen(InString); 1)
1200         If (StrPos(RefString; SubStr(InString; Count; 1))=0)
1201             Return (False)
1202         EndIf
1203     EndFor
1204     Return (True)
1205 EndFunc
1206 Label(NewNum) // this routine massages numeric values into new string values
1207 If(var=0)
1208     Return
1209 EndIf
1210 x=StrPos(var; "-")
1211 If(x=1)
1212     var=Substr(var; 2; strlen(var)) NegNum=1
1213 EndIf
1214 x=StrPos(var; ".")
1215 Switch(x)
1216     Caseof 0: tNum1=var Call(mainNum) tNum2=""
1217     Default: tNum1=Substr(var; 1; x-1) Call(mainNum) tnum2=substr(var; x; strlen(var))
1218         x=strlen(tnum2)-1
1219         If(x<varD and vFinal=1)
1220             Repeat
1221                 tnum2=tnum2+"0" x=strlen(tnum2)-1
1222             Until(x=varD)
1223         EndIf
1224 EndSwitch
1225 var=tNum1+tNum2
1226 If(NegNum=1)
1227     var="-"+var
1228 EndIf
1229 Discard(NegNum; tNum1; tNum2)
1230 Return
```

```
1231 Label(mainNum) // this routine creates commas in the main number from Label(NewNum)
1232 If(StrLen(tNum1)<=3)
1233 Return
1234 EndIf
1235 z=Integer(StrLen(tNum1)/3)
1236 newvar=""
1237 Repeat
1238 y=SubStr(tNum1;StrLen(tNum1)-2;3)
1239 tNum1=SubStr(tNum1;1;StrLen(tNum1)-3)
1240 If(StrLen(tNum1)>3)
1241 newvar=","+y+newvar Else newvar=tNum1+","+y+newvar
1242 EndIf
1243 If(StrLen(tNum1)>3)
1244 z=Integer(StrLen(tNum1)/3) Else z=0
1245 EndIf
1246 Until(z=0)
1247 tNum1=newvar
1248 Return
1249 Label(MakePer) // this routine creates percentage string values; revised 6/4/2004 for scientific notation values
1250 e=0
1251 var=StrTransform(var;";";"")
1252 var=strnum(var)
1253 var=var*100 // a % value is always 100 times the non-percentage value
1254 x=strpos(var;"e") // scientific notation routine
1255 If(x>0)
1256 e=1
1257 y=substr(var;x+2;3) y=strnum(y) y=y+1
1258 var=substr(var;1;x-1)
1259 var=var-". "
1260 varq=strlen(var)
1261 If(varq<y)
1262 Repeat
1263 var=var+"0"
1264 Until (strlen(var)=y)
1265 EndIf
1266 x=strpos(var;".")
1267 If(x=0)
1268 var=var-". "
1269 EndIf
1270 EndIf
1271 If(e=0)
1272 var=numstr(var) // want to determine if that following the decimal is greater than 0, but not if scientific notation
1273 EndIf
1274 x=strpos(var;".") testn=substr(var;x+1;strlen(var)) testn=strnum(testn) // testn is rem w/o the "."
1275 If(testn=0)
1276 var=substr(var;1;x-1) // if the remainder is 0, drops the ".0" from var
1277 EndIf
1278 If(e=0) // not if scientific notation
1279 var=strnum(var)
1280 EndIf
1281 Call(NewNum)
1282 var=var+"%"
1283 If(negNum=1)
1284 var="-"+var
1285 EndIf
1286 Return
```



```
1287 Label(DateCnv) // this tests and sets the numeric parts of a m/d/yyyy date and makes string values
1288 Call(dateChk)
1289 If(exists(q))
1290 Return
1291 EndIf
1292 x=strPos(var;"/")
1293 If(x>0)
1294 q=1
1295 vMsg=var+" is not a legal date. Don't use consecutive '/' marks." setResult Return
1296 EndIf
1297 x=StrPos(var;"/")
1298 If(x=0)
1299 q=1
1300 vMsg=var+" is not a legal date. Use a m/d/yyyy date format." setResult Return
1301 EndIf
1302 If(x>0)
1303 mo=substr(var;1;x-1) rem=substr(var;x+1;strlen(var))
1304 x=StrPos(rem;"/")
1305 day=substr(rem;1;x-1) year=substr(rem;x+1;strlen(rem))
1306 If(strlen(mo)>2 or strlen(day)>2 or strlen(year)<>4)
1307 q=1
1308 vMsg=var+" is not a legal date. Use a m/d/yyyy date format." Return
1309 EndIf
1310 EndIf
1311 Mo=Strnum(Mo) Day=Strnum(Day) Year=strnum(Year)
1312 If(year<1601)
1313 q=1
1314 vMsg="Dates earlier than 1/1/1601 cannot be used. Change the date" Return
1315 EndIf
1316 If(Mo=0 or Day=0 or Year=0)
1317 q=1
1318 vMsg="Don't use zeros for months, days or years. "+Mo+"/"+Day+"/"+Year+" is not a legal
1319 number."
1320 Return
1321 EndIf
1322 If(Mo>12)
1323 q=1
1324 vMsg=Mo+" is not a legal month. Use 1~12." Return
1325 EndIf
1326 Switch(Mo)
1327 Caseof 1: vMo="January" Caseof 7: vMo="July"
1328 Caseof 2: vMo="February" Caseof 8: vMo="August"
1329 Caseof 3: vMo="March" Caseof 9: vMo="September"
1330 Caseof 4: vMo="April" Caseof 10: vMo="October"
1331 Caseof 5: vMo="May" Caseof 11: vMo="November"
1332 Caseof 6: vMo="June" Caseof 12: vMo="December" EndSwitch
1333 vLeap=DatelsLeapYear(year) // this is used for determining what to do with February, below
1334 Switch(vMo)
1335 Caseof "February":
1336 Switch(vLeap)
1337 Caseof False:
1338 If(day>28)
1339 q=1
1340 EndIf
1341 Caseof True:
1342 If(day>29)
```

```
1343         q=1
1344     EndIf
1345 EndSwitch
1346 Caseof "April":
1347     If(Day>30)
1348         q=1
1349     EndIf
1350 Caseof "June":
1351     If(Day>30)
1352         q=1
1353     EndIf
1354 Caseof "September":
1355     If(Day>30)
1356         q=1
1357     EndIf
1358 Caseof "November":
1359     If(Day>30)
1360         q=1
1361     EndIf
1362 Default:
1363     If(Day>31)
1364         q=1
1365     EndIf
1366 EndSwitch
1367 If(Exists(q))
1368     vMsg=vMo+" "+year+" does not contain "+day+" days." Return
1369 EndIf
1370 var=vMo+" "+Day+", "+Year
1371 Return
1372 Label(dateChk) // this tests for m/d/yyyy string legal date characters
1373 ret=IsNumberDate(var)
1374 If(Ret)
1375     Else vMsg=var+" is not a legal date. " q=1
1376 EndIf
1377 Return
1378 Function IsNumberDate (InString)
1379     RefString="1234567890/"
1380     ForNext (Count;1;StrLen(InString);1)
1381         If (StrPos(RefString;SubStr(InString;Count;1))=0)
1382             Return (False)
1383         EndIf
1384     EndFor
1385     Return (True)
1386 EndFunc
1387 Label(vError) // this is the initial OnError() Label
1388 x=?NumberOpenDocuments
1389 If(x=9)
1390     vMsg="You have attempted to make a new blank document."+h+h+ "WordPerfect can only
1391     have 9 open documents at a time, and you are at that point. "+h+h vMsg=vMsg+"You may
1392     continue to use the macro, but no new documents can be opened at this time. "+h+h+"To
1393     stop the macro, click the Stop button."
1394     Else
1395     vMsg="An error has occurred for unknown cause. "+h+h vMsg=vMsg+"If you receive this
1396     message, please send me an e-mail (loudenbk@swbell.net) or call me (405-236-4004) "
1397     vMsg=vMsg+"and describe what you were doing when the error occurred and I'll try to fix the
1398     problem."+h+h vMsg=vMsg+h+"It will help to know your WordPerfect version and your
```

```

1399   operating system, e.g., WindowsXP."+h+"The macro must now stop."
1400 EndIf
1401 DialogShow("verr";"WordPerfect") Return
1402 // End of Math.wcm

```

Top of Chapter 9

Top of Macro

Chapter 10

```

1  //CONVERTFE.WCM
2  // © 1998-2004 by Doug Loudonback, Oklahoma City, All Rights Reserved
3  //This macro may be distributed or edited, but may not be sold or posted on any website without my
4  express written permission.
5  //Revision Date: 6/11/2004
6  //   This macro converts either endnotes or footnotes to plain text "pseudo" endnotes at the end of the
7  document, but preserves the appearance of endnotes – note numbers remain in superscript.
8  // This macro is downloadable at http://www.dogloudenback.com/wp/convertFE.exe
9  Application (WordPerfect; "WordPerfect"; Default!; "EN")
10 H=NTOC(0F90Ah) Display(On!)
11 If(?docBlank)
12   vMsg="This is a blank document. It contains no endnotes or footnotes so there's nothing for
13   this macro to do."
14   DialogShow("NoNotes2";"WordPerfect") Quit
15 EndIf
16 BookmarkCreate("x") // a Bookmark is created at the top; it will later be deleted
17 fsz=?FontSize // this is the font name at the top of the document
18 vfont=?font // this is the font size at the top of the document
19 fsz=ConvertType(fsz;Points!) fsz=ConvertType(fsz;String!)
20 fsz=substr(fsz;1;strlen(fsz)-1) vsz=fsz
21 vLoop=False
22 DialogShow("vType";"WordPerfect";cbType)
23 vtxt="1. Text will wrap in paragraphs"+h+"   like this."
24 RegionSetWindowText("vType.S0";vtxt)
25 Repeat Until(vLoop) DialogDestroy("vType")
26 If(x=2) // Cancel Button is pushed in the dialog ...
27   BookmarkFind("x") BookmarkDelete("x") Quit
28 EndIf
29 Switch(vWhat)
30   Caseof "Endnotes": var="[Endnote]" // [Endnote] and [Footnote] are not text
31   Caseof "Footnotes": var="[Footnote]" // they are inserted using the Codes ... button on the Macro Toolbar
32 EndSwitch
33 Display(Off!)
34 Label(vTest) // this tests the document to see if it contains what you wanted to convert - Endnotes or Footnotes
35 PosDocTop // moves the insertion point to the top of the document
36 OnNotFound(NoNotes) // what happens if the document contains no notes
37 SearchString (var) MatchPositionAfter SearchNext
38 // the "OnNotFound" condition did not occur; the macro continues
39 PosDocBottom SearchString(var) MatchPositionAfter SearchPrevious
40 Switch(vWhat) // new on 6/11/04; gets last note number value for progress dialog (vWait)
41   Caseof "Endnotes":
42     y=?EndNote y=numstr(y) // gets last endnote value
43   Caseof "Footnotes":
44     y=?FootNote y=numstr(y) // gets last footnote value
45 EndSwitch
46 PosDocTop
47 Main=?DocNumber // this identifies the source document being worked on
48 FileNew Notes=?DocNumber // this identifies the temporary document containing pseudo notes
49 QuickCorrectQuickBulletsSet(Off!) //these 2 items were added 6/11/04;
50 QuickCorrectQuickIndentSet(Off!) //if "on", printing issues are present

```

```

51 Font(vFont) FontSize(strnum(vSz) + "p") // font size and other characteristics of the Notes document is set
52 ParagraphSpacing (vParSp) // revised 6/11/04: replaces former code with variable paragraph spacing
53 TabSet (Origin: Relative!) //revised 6/11/04: allows for tab/indent of ¶ numbers
54 TabSet (Relative!; { 1.0"; TabLeft!; 1.38"; TabLeft!; 1.6"; TabLeft!; 1.9"; TabLeft!; 2.2";
55 TabLeft!; 2.5"; TabLeft!; 2.8"; TabLeft!; 3.1"; TabLeft!; 3.4"; TabLeft!; 3.7"; TabLeft!;
56 4.0"; TabLeft!; 4.3"; TabLeft!; 4.6"; TabLeft!; 4.9"; TabLeft!; 5.2"; TabLeft!; 5.5";
57 TabLeft!; 5.8"; TabLeft!; 6.1"; TabLeft!; 6.4"; TabLeft!; 6.7"; TabLeft!; 7.0"; TabLeft!;
58 7.3"; TabLeft!; 7.6"; TabLeft!; 7.9"; TabLeft!; 8.2"; TabLeft!})
59 HLineCreate HardReturn Advance(AdvanceDown!; 0.101")
60 Center Type("NOTES") HardReturn BookmarkCreate("x")
61 SwitchDoc(Main)
62 Msg1="Finding and converting notes" Msg2="Note "+x+" of "+y+" Please wait . . ."
63 Call(WaitMsg)
64 Label(Begin)
65 OnNotFound(vNext) // what happens if no more notes are found
66 SearchString (var) MatchPositionAfter SearchNext
67 Switch(vWhat)
68   Caseof "Endnotes":
69     x=?EndNote x=numstr(x) EndnoteEdit(x) // sets found note value and enters it for copying
70   Caseof "Footnotes":
71     x=?FootNote x=numstr(x) FootnoteEdit(x) // sets note value and enters it for copying
72 EndSwitch
73 RegionSetWindowText("vWait.S0";"Note "+x+" of "+y+" Please wait . . .") //new on 6/11/04:
74 shows progress
75 PosDocTop
76 Repeat //new on 6/11/04: makes sure leading "spaces" in the note are eliminated for copy/paste routine
77   c=?RightChar
78   If(c=" ")
79     PosWordNext EndIf
80   Until(c<>" ")
81 Repeat //new on 6/11/04: makes sure leading "LeftTab" codes in the note are eliminated for copy/paste routine
82   c=?RightCode
83   If(c=4576)
84     PosWordNext EndIf
85   Until(c<>4576)
86 // the code below copies a note's text, exits the note, switches to the Notes doc, pastes, and returns to the main doc
87 SelectMode(On!) // this positions the insertion point at the top of the note and turns on Select
88 PosDocBottom // this positions the insertion point at the bottom of the note
89 EditCopy SelectMode(Off!) Close // this copies the selection to Windows Clipboard and exits the note
90 SwitchDoc(Notes) PosDocBottom Type(x + ".") //switches to the "notes" document, types note #
91 If(vIndent="Tab") Tab Else Indent EndIf //new on 6/11/04: uses Tab or Indent per user choice
92 EditPaste PosLineEnd HardReturn
93 // the above switches to the temporary Notes document, types a "note" number, and pastes the Clipboard value
94 SwitchDoc(Main) Go(Begin) // the search loops and continues until no notes are found
95   Label(vNext) // if no more notes are found, the macro continues here
96 SwitchDoc(Main) PosDocTop
97   Label(vNext2) //routine loops until no more notes are found
98 OnNotFound(vDone) SearchString (var) MatchPositionAfter SearchNext
99 Switch(vWhat)
100   Caseof "Endnotes": x=?EndNote x=numstr(x)
101   Caseof "Footnotes": x=?FootNote x=numstr(x)
102 EndSwitch
103 FontSuperscriptToggle HardSpace Type(x) FontSuperscriptToggle Go(vNext2)
104 Label(vDone) // these routines clean up the 2 documents, deleting all true footnotes or endnotes
105 PosDocTop // in the original document here
106 SearchString (var) ReplaceString ("") ReplaceForward (Extended!)

```

```

107 SwitchDoc(Notes) // in the Notes document here
108 BookMarkFind("x")
109 SearchString("[HRT][HRT]") ReplaceString("[HRT]") ReplaceForward //insert [HRT] with Codes... Bttn
110 BookMarkFind("x")
111 SearchString("[Font Size]") ReplaceString("") ReplaceForward //insert [Font Size] code with Codes... Bttn
112 BookMarkFind("x")
113 SearchString("[Tab Set]") ReplaceString("") ReplaceForward //insert [Tab Set] code with Codes... Bttn
114 BookMarkFind("x") BookMarkDelete("x")
115 // this copies the temporary Notes document and pastes it at the end of the original document
116 PosDocTop
117 PosDocVeryTop SelectMode(On!) PosDocBottom
118 EditCopy SelectMode(Off!) CloseNoSave
119 SwitchDoc(Main)
120 PosDocBottom HardReturn EditPaste
121 BookmarkFind("x") BookmarkDelete("x")
122 KillMsg // stops the progress dialog
123 Msg1="The operation is done." Msg2="To undo changes, click your Undo icon"
124 WaitMsg Wait(20) KillMsg
125 Quit
126 Label(NoNotes) // this sees if the original "type" of notes are in the main document; if not, it the other type IS
127 OnNotFound(vNone)
128 PosDocTop Switch(vWhat)
129 Caseof "Endnotes": var="[Footnote]" // insert the [Footnote] code with Codes... button
130 Caseof "Footnotes": var="[Endnote]" // insert the [Endnote] code with Codes... button
131 EndSwitch
132 SearchString(var) MatchPositionBefore SearchNext
133 Switch(vWhat)
134 Caseof "Endnotes":
135     vmsg="This document contains Footnotes, not Endnotes. Do you want to do Footnotes
136     instead?"
137 Caseof "Footnotes":
138     vmsg="This document contains Endnotes, not Footnotes. Do you want to do Endnotes
139     instead?"
140 EndSwitch
141 Display(On!)
142 DialogShow("NoNotes"; "WordPerfect") x=macrodialogresult DialogDestroy("NoNotes")
143 If(x="CancelBttn" or x=2) BookmarkFind("x") BookmarkDelete("x") Quit EndIf
144 Display(Off!)
145 Switch(vWhat)
146 Caseof "Footnotes": vWhat="EndNotes" var="[Endnote]" // insert the [Endnote] with Codes... button
147 Caseof "Endnotes": vWhat="Footnotes" var="[Footnote]" // insert the [Footnote] with Codes... button
148 EndSwitch
149 Go(vTest)
150 Label(vNone) // neither Footnotes or Endnotes are present in the main document
151 Display(On!)
152 vmsg="This document contains neither Footnotes nor Endnotes. So, there is nothing for it to do
153 and the macro will stop." DialogShow("NoNotes2"; "WordPerfect") DialogDestroy("NoNotes2")
154 BookmarkFind("x") BookmarkDelete("x") Quit
155 Label(cbType) Switch(cbType[3]) // this is the callback label for/from the main dialog
156 Caseof "OKBttn"; 1: vClose
157 Caseof "B2": //new routine on 6/11/04
158     var=RegionGetSelectedText("vType.B2") //changes "vType.S0" control to show effect of selection
159     If(var="Tab")
160         vtxt="1. Text will wrap in paragraphs"+h+"like this."
161     Else
162         vtxt="1. Text will wrap in paragraphs"+h+" like this."

```

```

163 EndIf
164 RegionSetWindowText("vType.S0";vtxt)
165 Caseof "CancelBtn"; 2: vClose
166 Caseof "TipsBtn": DialogShow("vTips";"vType")
167 EndSwitch Return
168 Label(vClose) //closes the callback dialog
169 DialogDismiss("vType";cbType[3]) x=MacroDialogResult vLoop=True
170 Return
171 Label(WaitMsg) DialogShow("vWait";"WordPerfect";cbWait) Return
172 Label(cbWait) If(cbWait[3]="CancelBtn") Assert(CancelCondition!) EndIf Return
173 Label(KillMsg) DialogDismiss("vWait";2) DialogDestroy("vWait") Return
174 // end of macro

```

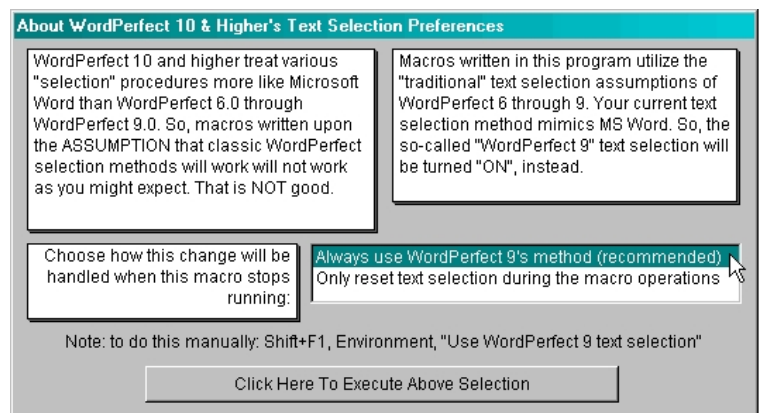
[Top of Chapter 9](#)

[Top of Macro](#)

[Chapter 10](#)

Wp9select.wcm. It exists for a single purpose – to deal with the text selection default used in WordPerfect 10 and higher. See the discussion in [Chapter 3](#) and [BMT](#) for other means of handling the problem. Download this macro at <http://www.dougloudenback.com/wp/wp9select.exe>.

When the 1st part of the macro code is run in WordPerfect 10 or higher and the "Environment" Preference for "Use WordPerfect 9 text selection" is not checked, this dialog will appear. The user can "leave" WordPerfect 9 text selection turned "On" or can turn it off when the macro(s) are done by using the 2nd section of the code in this or some other related macro.



```

1 //WordPerfect Macro: wp9select.wcm
2 //This identifies and sets Preferences in Wp10 through 12 for Wp9 text selection: gives model text
3 //7/2/2004: I've just learned from JDan Broadhead that platform ID's can be combined in the
4 //IfPlatform statement, so that instead of a separate statement for each (_VersionX!), a single
5 //statement could read like this:
6 //IfPlatform(_Version10!; _Version11!; _Version12!)
7 // x=?Wp9Selection
8 // If(x=0)
9 // selChange=1 ChangeSel
10 // PrefEnvironment (; WP9Selection: On!) PrefSave
11 // EndIf
12 //EndIfPlatform
13 //I'll leave the code "as is", but it would be simpler to combine the Platform ID's, as shown at Line 6, above
14
15 Application (WordPerfect; "WordPerfect"; Default!; "EN") VarErrChk(Off!)
16 vSelText=0 selChange=0 ChangeSel // this avoids a compilation warning if run under Wp 6.1, 7, 8 or 9
17
18 IfPlatform(_Version10!) // will compile only in WordPerfect 10
19 x=?Wp9Selection
20 If(x=0)
21 selChange=1 ChangeSel
22 PrefEnvironment (; WP9Selection: On!) PrefSave
23 EndIf
24 EndIfPlatform
25
26 IfPlatform(_Version11!) // will compile only in WordPerfect 11
27 x=?Wp9Selection

```

```
28     If(x=0)
29         selChange=1 ChangeSel
30         PrefEnvironment (; WP9Selection: On!) PrefSave
31     EndIf
32 EndIfPlatform
33
34 IfPlatform(_Version12!) // will compile only in WordPerfect 12
35     x=?Wp9Selection
36     If(x=0)
37         selChange=1 ChangeSel
38         PrefEnvironment (; WP9Selection: On!) PrefSave
39     EndIf
40 EndIfPlatform
41
42 // in a "real" macro, the following code would be used to reset Wp9 text selection in Wp 10, 11 & 12, IF
43 variable vSelText exists and =1; in this macro, by itself & under the stated conditions, Wp9 text selection
44 will remain "Off" unless the following code is present
45 If(Exists(vSelText) and vSelText=1)
46
47     IfPlatform(_Version10!) // will compile only in WordPerfect 10
48         PrefEnvironment (; WP9Selection: Off!) PrefSave
49     EndIfPlatform
50
51     IfPlatform(_Version11!) // will compile only in WordPerfect 11
52         PrefEnvironment (; WP9Selection: Off!) PrefSave
53     EndIfPlatform
54
55     IfPlatform(_Version12!) // will compile only in WordPerfect 12
56         PrefEnvironment (; WP9Selection: Off!) PrefSave
57     EndIfPlatform
58 EndIf
59 Return
60 Label(ChangeSel)
61 If(selChange=0) // this avoids a compilation warning if run under Wp 6.1, 7, 8 or 9
62     Else
63         DialogShow("vSelect"; "WordPerfect")
64         vSelText=Substr(vSelText; 1; 1)
65         If(vSelText="O") // in this or a related macro, this will be used to turn off Wp9 text selection
66             Discard(vSelText) Global vSelText=1 // this discards Local variable vSelText & sets Global
67         Else
68             Discard(vSelText) // no variable vSelText will exist
69         EndIf
70 EndIf
71 Return
```


NOTES

NOTES

NOTES

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	<p>"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).</p> <p>A boolean value is <i>True</i> (for practical purposes, the same as 1) or <i>False</i> (for practical purposes, the same as 0).</p> <p>Sometimes, WordPerfect macros return a value with the non-string word <i>True</i> or <i>False</i>, 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:</p> <pre>x=QuickCorrectQuickBulletsQry // obsolete in Wp10 but it will compile MessageBox(;"x";x) // x will equal 1 x=MacrolsCompiled("allfonts.wcm") //if Shipping Macros are installed</pre>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<pre>Messagebox(;"x";x) // x will equal True If(x=1) Messagebox(;"1 is same as True";"") EndIf</pre>
General Note About the := sign	Wp8 & higher	<p>:= 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:</p> <pre>var := numeric * numeric</pre> <p>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.</p> <p>So, var=numeric*numeric works every bit as well as</p> <pre>var := numeric*numeric.</pre> <p>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.</p>
General Note Rationale for including or excluding stuff in this book	All	<p>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.</p>
General Note Parameters and Use of ()	All	<p>() 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.</p> <p>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.</p>
General Note Use of "" for String Values	All	<p>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.</p> <p>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.</p> <p>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.</p> <p>See ?QuickCorrect and the various QuickCorrect items, below.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
<p>General Note</p> <p>Don't Use "" for numeric, boolean or other non-string values</p>	All	<p>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.</p> <p>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.</p>
<p>General Note</p> <p>Use of VAR in this document</p>	All	<p>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.</p>
<p>General Note</p> <p>Use of Upper/Lower Case</p>	All	<p>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.</p>
<p>General Note</p> <p>Length of Single Code Line</p>	All	<p>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.</p>
<p>General Note</p> <p>Reserved Words</p>	All	<p>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, Ondeadvise, 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.</p> <p>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).</p>

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	<p>An operator; Usage: var=numeric*numeric</p> <p>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).</p> <p>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.</p> <p>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.</p>
/	All	<p>An operator; Usage: var=numeric / numeric</p> <p>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.</p> <p>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.</p>
+	All	<p>An operator; Usage: (1) var=numeric + numeric or (2) var=string+string</p> <p>(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.</p> <p>(2) String: The values will be combined. Also, see concatenation.</p> <p>Examples:</p> <p>x=500 y=1 z=x+y Result: z=501 (numeric value)</p> <p>x=500 y="1" z=x+y Result: z=501 (numeric value)</p> <p>x=500 y="\$1" z=x+y Result: z=500\$1 (string value)</p> <p>x="abc" y="def" z=x+y Result: z=abcdef (string value)</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
-	All	<p>An operator; Usage: (1) var=numeric - numeric or (2) var=string-string</p> <p>(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).</p> <p>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.</p> <p>(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.</p> <p>According to J. Dan:</p> <p>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.</p> <p>var1="abcdef" var2="bcd" var3=var1-var2</p> <p>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.</p>
=	All	<p>Usage: var=any = any</p> <p>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:</p> <p>If(varA=varB) [do this] Else [do that] EndIf</p> <p>Either string, numeric, or Boolean values may be compared.</p>
variable = or variable :=	All	<p>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.</p> <p>Examples: var=var5 var=5 var="Father"</p> <p>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.</p> <p>See Reserved Words, above, and/or Variables, below.</p>
<> or !=	All	<p>Usage: var=any <> any</p> <p>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:</p> <p>If(varA <> varB) [do this] Else [do that] EndIf</p> <p>Either strings or numerical values may be compared.</p> <p>Examples if var1 = "Father" and var2 = "father":</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>If(var1=var2) MsgBox("Values are the same");") Else MsgBox("Values are not the same");") EndIf</p> <p>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:</p> <p>If(ToLower(var1) = ToLower(var2)) MsgBox("Values are the same");") Else MsgBox("Values are not the same");") EndIf</p>
<	All	<p>Usage: var=any < any</p> <p>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:</p> <p>If(varA < varB) [do this] Else [do that] EndIf</p> <p>Either string or numeric values may be compared. If strings are compared, True is returned if the left value alphabetically precedes the right value.</p>
<=	All	<p>Usage: var=any <= any</p> <p>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:</p> <p>If(varA <= varB) [do this] Else [do that] EndIf</p> <p>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.</p>
>	All	<p>Usage: var=any > any</p> <p>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:</p> <p>If(varA > varB) [do this] Else [do that] EndIf</p> <p>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" MsgBox("c");c) //c=True</p>
>=	All	<p>Usage: var=any >= any</p> <p>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:</p> <p>If(varA >= varB) [do this] Else [do that] EndIf</p> <p>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" MsgBox("c");c) //c=True.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
<p style="text-align: center;">System Variables</p> <p>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.</p>		
?DateDay	All	<p>Usage: var=?DateDay</p> <p>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.</p>
?DateMonth	All	<p>Usage: var=?DateMonth</p> <p>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.</p>
?DateWeekDay	Wp 8 & higher	<p>Usage: var=?DateWeekday</p> <p>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.</p>
?DateYear	All	<p>Usage: var=?DateYear</p> <p>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.</p>
?DisplayMode	All	<p>Usage: var=?DisplayMode</p> <p>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.</p>
?DocBlank	All	<p>Usage: var=?DocBlank</p> <p>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.</p>
?EndNote	All	<p>Usage: var=?EndNote</p> <p>Assigns a numeric value to var, that being the number of the endnote to the left of the insertion point. Also, see ?Footnote, below.</p>
?Font	All	<p>Usage: var=?Font</p> <p>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.</p>
?FontSize	All	<p>Usage: var=?FontSize</p> <p>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.</p>

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 Files (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 value 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
<p style="text-align: center;">All The Rest</p> <p>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.</p>		
/* <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	<p>Usage: var=boolean AND boolean</p> <p>Assigns var the boolean (True/False) value if all the compared expressions are True, False if not.</p> <p>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:</p> <p>If(var1>500 AND var2=50) [do this] Else [do that] EndIf</p>
OR	All	<p>Usage: var=boolean OR boolean</p> <p>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.</p> <p>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:</p> <p>If(var1>500 OR var2=50 OR var3=25) [do this] Else [do that] EndIf</p>
Appearance attributes	All	For appearance attributes, e.g., Bold, see Attribute commands , below.
Application()	All	<p>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.</p> <p>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.</p> <p>For WordPerfect 6.1 and 7.0 USA editions, the Application statement would read (if you created a macro from your keyboard):</p> <p style="padding-left: 40px;">Application(A1; "WordPerfect"; Default; "US")</p> <p>For WordPerfect 8.0 & higher, if you create a macro from the keyboard, the statement would read:</p> <p style="padding-left: 40px;">Application (WordPerfect; "WordPerfect"; Default!; "EN")</p> <p>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.</p> <p>By way of further explanation, J. Dan Broadhead notes that:</p> <p style="padding-left: 40px;">The "A1" thing is a bit misleading.</p> <p style="padding-left: 40px;">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</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>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).</p> <p>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.</p> <p>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:</p> <p>Application (WP; "WordPerfect"; Default)</p> <p>In fact, even Default is optional, so this would work too:</p> <p>Application (WP; "WordPerfect")</p>
Arrays	All	See Declare() , below.
Assert()	All	<p>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.</p> <p>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.</p>
AttributeAppearanceOff () AttributeAppearanceOn () AttributeAppearanceToggle ()	All	<p>Turns off/on the following format codes, which you'd include within the (): Bold!; DoubleUnderline!; Every!; Italics!; Outline!; Redline!; Shadow!; SmallCaps!; Strikeout!; Underline!</p> <p>Separate multiple attributes with semicolon(s):</p> <p>AttributeAppearanceOff(Bold!) AttributeAppearanceOn(Bold!; Italics!)</p> <p>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.</p> <p>Example: AttributeAppearanceToggle (Underline!)</p>
AttributeNormal	All	Usage: As shown in the left column; turns off all current font attributes except color.
AttributePosition() AttributePositionToggle()	All	<p>Use as shown in the left column, adding one of the following:</p> <p>NormalPosition! or Subscript! or Superscript!</p> <p>Example: AttributePosition(Superscript!)</p> <p>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.</p> <p>Example: AttributePositionToggle (Normal!)</p>
AttributeRelativeSize() AttributeRelativeSizeToggle()	All	<p>Use as shown in the left column, adding one of the following:</p> <p>ExtraLarge! or Fine! or Large! or NormalSize! or Small! or VeryLarge!</p> <p>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.</p> <p>Example: AttributeRelativeSizeToggle (Large!)</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
Average()	Wp 8 & higher	<p>Usage: var=Average (numeric values separated by semicolons)</p> <p>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.</p> <p>Example: var=Average (2; 4; 6; 8) – the value of var is 5.0</p> <p>Example: var=Average (32.53333; 59.122334567; 33.33; 49.22) – the value of var is 43.55141614175</p> <p>You can use other commands to round the result if you wish, e.g.,</p> <p>var=Average (32.53333; 59.122334567; 33.33; 49.22)</p> <p>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).</p>
Beep	All	<p>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:</p> <p>Default! (used if no parameter is specified) or Question! or Asterisk! or Exclamation! or CriticalStop!</p> <p>So, with a parameter, the statement might read: Beep(Question!); without a parameter, just use Beep, without the ().</p> <p>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.</p>
BlockProtect()	All	<p>Usage: BlockProtect(On!) and/or BlockProtect(Off!)</p> <p>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.</p>
BookmarkCreate(string)	All	<p>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>.</p> <p>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.</p>
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]		<p>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. <i>If a Return command is not present</i>, 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:</p> <p>-----</p> <p>Type("This line of type is being typed into a new document.") Call(H)</p> <p>Type("This is the second line of type being typed into the same document.") Call(H)</p> <p>Quit</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>Label(H) HardReturn HardReturn Return Type("Donald Duck is dumb.") -----</p> <p>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.</p> <p>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".</p> <p>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 ...</p> <p>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.</p> <p>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.</p> <p>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.</p>
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	<p>Inserts a [HdCenteronMarg] code at the insertion point, centering the current line of text, e.g.,</p> <p>Type("Chapter One") Center Type("Part B") FlushRight DateCode HardReturn</p>
Chain()	All	<p>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."</p> <p>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.</p>

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
		<p>DatePart (WordPerfect 8.0 & higher)</p> <p>DateString (WordPerfect 7.0 & higher)</p> <p>DateWeekday (WordPerfect 7.0 & higher)</p> <p>DateWeekdayName (WordPerfect 7.0 & higher)</p> <p>DateWeekOfYear (WordPerfect 8.0 & higher)</p> <p>DateYear (WordPerfect 7.0 & higher)</p> <p>See Chapter 7, Date Routines.</p>
DDE Commands	All	<p>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.</p>
Declare var[num]	All	<p>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".</p> <p>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.</p> <p>Usage if a one dimensional array: Declare var[num]</p> <p>Usage if a two dimensional array: Declare var[num1;num2]</p> <p>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:</p> <p>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.</p> <p>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 misled by this example – an element can be assigned numeric values, formulas, etc., even though strings (text) is used here.</p> <p>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.</p>
DeleteCharNext DeleteCharPrevious DeleteToEndOfLine DeleteWord	All	<p>Deletes the character (or, if something is selected, the selection – SelectDelete does the same) to the right of the insertion point.</p> <p>Deletes the character (or, if something is selected, the selection) to the left of the insertion point. Also, see SelectDelete, below.</p> <p>Deletes text and codes from the insertion point to the end of the line.</p> <p>Deletes the current word or space if the insertion point is not in a word.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
DialogAdd...	All	<p>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.</p> <p>Consult your WordPerfect on-line Macro help for information about DialogDefine() and the various DialogAdd... statements which are possible in that context.</p> <p>For examples, see DialogShow w/CallbackWait, MacroCompile, Copying a Dialog to Text in Chapter 5, and Sample RegionGet Code in Chapter 8.</p>
DialogDefine()	All	<p>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.</p>
DialogDestroy()	Wp7.0 & higher	<p>Removes a Dialog from memory. The following comments relate to DialogDestroy 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.</p> <p>WordPerfect 6.1: Don't use the command.</p> <p>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".</p> <p>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.</p> <p>Usage. After the DialogShow() command, (a) if you need to know the value of the control which closes the dialog (e.g., "OKBtn", "CancelBtn", "OtherBtn") 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 "OKBtn" control), you don't need to bother with using MacroDialogResult since its value doesn't matter.</p> <p>Example: DialogShow("vTest"; "WordPerfect") x=MacroDialogResult DialogDestroy("vTest"); variable x will be the value of the button that closed the dialog (1 for an "OKBtn"; 2 for a "CancelBtn" or a Windows "Close" (x) button, or, otherwise, the string name of the control, e.g., "HelpBtn"). Then, you could write: If(x=1) [do this] Else [do that] EndIf.</p> <p>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).</p> <p>If using a dialog with a callback function: Same as above, except use the command after DialogDismiss().</p> <p>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.</p>
DialogDismiss()	Varies	<p>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</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>read the substantial discussion in Chapter 8 before deciding what to do.</p> <p>If a DialogShow() command does NOT include a Callback function:</p> <p>WordPerfect 6.1: Although contrary to orthodoxy, and, so, not recommended, I found that DialogDismiss("Help1"; 1) did no harm.</p> <p>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.</p> <p>All Versions, if a DialogShow() command includes a Callback routine: Use DialogDismiss() either after or within the Callback function itself.</p> <p>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:</p> <pre> 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 </pre> <p>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.</p> <p>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.</p> <p>Here's the better way (shown more completely in Example 1, Chapter 8):</p> <pre> 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 </pre> <p>So, what's the difference, and what was that 1 in the 1st method which started this comparison?</p> <p>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).</p> <p>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</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>MacroDialogResult would be when used following the callback. That was a pragmatic decision, not a necessary one. The statement, <code>If(x=2) Quit EndIf</code>, could just as well have been, <code>If(MacroDialogResult=2) Quit EndIf</code>. Either way, the use of <code>vQuit</code> is no longer necessary and <code>DilaogDismiss()</code> no longer uses an arbitrary, if not fictitious, value to hide the dialog.</p> <p>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 "CancelBtn" 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 <code>vQuit=False</code>, which is not necessarily always the case).</p> <p>See the discussion in DialogDestroy(), above. Also, see DialogShow() [not using <code>CallbackWait</code>] and DialogShow() [using <code>CallbackWait</code>] and this substantial discussion in Chapter 8 of this and related topics.</p>
<code>DialogDisplay()</code>	All	<p><code>DialogDisplay()</code> is an obsolete command. Use DialogShow() [not using <code>CallbackWait</code>] or DialogShow() [using <code>CallbackWait</code>].</p>
<p><code>DialogShow()</code> [not using <code>CallbackWait</code> and <code>CallbackResume</code>] For All Wp Versions</p>		<p>This discussion is not applicable if you want to use the <code>CallbackWait</code> <code>Callback Resume</code> commands present in WordPerfect 7 & higher. For that, see DialogShow() [using <code>CallbackWait</code>], 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.</p> <p>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:</p> <p>No Callback: <code>DialogShow("vDialog"; "WordPerfect")</code></p> <p>Concerning use of <code>DialogShow</code> with or without callbacks, see Chapter 8.</p> <p>Concerning statements after/before the <code>DialogShow()</code> command, see DialogDismiss and MacroDialogResult and the substantial discussion about such matters in Chapter 8.</p> <p>With Callback: <code>DialogShow("vDialog"; "WordPerfect"; cbvDialog)</code>, where <code>vDialog</code> 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.</p> <p>In this regard, when a callback routine is used, the <code>DialogShow()</code> command is typically preceded by one or both of these commands: <code>vLoop=False vQuit=False</code>, but see this better approach in DialogDismiss(), above. In the callback label, at the point that <code>vLoop</code> is redefined as <code>vLoop=True</code>, the callback loop is flagged to stop. If <code>vQuit</code> has been redefined as <code>vQuit=False</code>, 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.</p> <p>After the <code>DialogShow()</code> statement, a <code>Repeat Until(vLoop)</code> statement is commonly used which instructs the macro how long the loop will last. Notice the <code>Repeat Until(vLoop)</code> statement below. The <code>Repeat</code> statement can be preceded by other statements relating to the looping dialog, e.g.,</p> <pre>If(Exists(var9)) RegionShowWindow("vDialog.B4"; Show!) Else RegionShowWindow("vDialog.B4"; Hide!) EndIf</pre> <p>In this regard, see Commands At The Beginning of Dialog Show in Chapter 8.</p> <p>When the <code>Until (vLoop)</code> 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:</p> <pre>vLoop=False vQuit=False DialogShow("vDialog"; "WordPerfect"; cbvDialog)</pre>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>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]</p> <p>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</p> <p>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.</p>
<p>DialogShow() [using CallbackWait and CallbackResume]</p> <p>For WordPerfect 7.0 & higher</p>		<p>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.</p> <p>WordPerfect's Macro help says, "CallbackWait and CallbackResume are easier to use and more efficient than loops." The principal structure is this:</p> <pre> 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] </pre> <p>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.</p> <pre> 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 </pre>

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)	All	Creates a new directory. Include the full pathname in quotation marks, e.g., DirectoryCreate("C:\Form Files")
DirectoryExists(var;string)		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	<p>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.</p> <p>Common usage: <code>x=Exists(var)</code></p> <p>Other usages: <code>x=Exists(var;Local!)</code> or <code>x=Exists(var;Global!)</code> or <code>x=Exists(var;Persistent!)</code></p> <p>Examples:</p> <p><code>x=Exists(var)</code> Result: If variable "var" exists, depending on whether it is a local, global, or persistent variable, <code>x=1</code> (local), <code>2</code> (global) or <code>3</code> (persistent), but if the variable does not exist, <code>x=0</code> (same as <code>NotFound!</code>) (numeric).</p> <p><code>x=Exists(var;Local!)</code> or <code>x=Exists(var;Global!)</code> or <code>x=Exists(var;Persistent!)</code> Result: If the type of variable enumerated exists, <code>x=True</code>, but, if not, <code>x=False</code> (boolean)</p> <p>Most often, an <code>Exists()</code> statement is combined with an If - EndIf statement or an If - Else - EndIf statement, e.g.,</p> <p><code>If(Exists(var)) [do this] Else [do that] EndIf</code></p> <p>Also, see Repeat/Until and While/Endwhile, below.</p>
FileExists(var;string)	All	<p>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:</p> <p><code>FileExists(var;"C:\Form Files\Petition.wpd")</code> will assign var a value of True if it exists, False if it does not. See FileInsert, FileOpen, below.</p>
FileInsert(string; enum; enum)	All	<p>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.</p> <p>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., <code>vPath="C:\Grande5\"</code>.</p> <p>If the specified file does not exist in the specified directory, an error will occur. Use FileExists() to determine that fact before the <code>FileInsert()</code> command is used, or use an error trapping routine - OnError() before using the command.</p> <p>Usage: <code>FileInsert(?PathDocument+"vfile.wpd")</code> or <code>FileInsert("C:\Grande5\vfile.wpd")</code> or <code>FileInsert(vPath+"vfile.wpd")</code> or <code>FileInsert("C:\Grande5\vfile.wpd"; Yes!; Insert!)</code> or <code>FileInsert("C:\Grande5\vfile.wpd"; Yes!; Prompt!)</code>. Also, see FileOpen(), below, and ?NumberOpenDocuments, above.</p>
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	<p>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.</p> <p>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\".</p> <p>If the specified file does not exist in the specified directory, or if 9 WordPerfect 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.</p> <p>Usage: FileOpen(?PathDocument+"vfile.wpd") or FileOpen("C:\Grande5\vfile.wpd") or FileOpen(vPath+"vfile.wpd") Also, see ?NumberOpenDocuments and FileInsert(), above.</p>
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	<p>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.</p> <p>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.</p>
FontSize()	All	<p>Sets font size at the insertion point. Use inch measurements. For example:</p> <p>FontSize (0.194") is 14p</p> <p>FontSize (0.167") is 12p</p> <p>FontSize (0.138") is 10p</p> <p>The simplest way to set font size is to use the Macro Toolbar's Record button, described in Chapter 5.</p>
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	<p>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 "I" (L) instead of a 1 or an O (Oh) instead of a 0 (zero), subsequently</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>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.</p> <p>See examples in Chapter 6, Chapter 7, and Chapter 9.</p>
GetNumber()	All	<p>Displays a dialog that lets you input a number in an edit box. The syntax is:</p> <p>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.</p> <p>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.</p> <p>See, by analogy, the GetString example in Chapter 4.</p>
GetString()	All	<p>Displays a dialog that lets you input a string in an edit box. The syntax is:</p> <p>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.</p> <p>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.</p> <p>See an example Chapter 4 for an example.</p>
Go()	All	<p>This command instructs macro operation to "go to" a particular Label in the macro. See Chapter 4.</p>
GraphicsLineCreate	All	<p>Creates a graphics line at the insertion point, using one of two possible parameters:</p> <p>GraphicsLineCreate(HorizontalLine!) or GraphicsLineCreate(VerticalLine!)</p> <p>Alternatively, use HLineCreate or VLineCreate.</p>
Graphics Commands	All	<p>Numerous other graphics commands are available but are not covered in this manual. See on-line Macro Help or other references.</p>
HardPageBreak	All	<p>Inserts a hard page break at the insertion point.</p>
HardReturn	All	<p>Inserts a hard return [HRT] at the insertion point. See NTOC, below.</p>
HardSpace	All	<p>Inserts a hard space [HSpace] at the insertion point, having the effect of holding adjacent words/characters together on the same line.</p>
Header Commands	All	<p>Numerous header commands are available but are not covered in this manual. See on-line Macro Help or other references.</p>
Help Commands	All	<p>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.</p>
HLineCreate	All	<p>Inserts a horizontal line from the left margin to the right margin. Also, see VLineCreate, below.</p>
Hypertext Commands	All	<p>Numerous hypertext commands are available but are not covered in this manual. See on-line Macro Help or other references.</p>
Hyphen	All	<p>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</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>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.</p> <p>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.</p>
If - Else - EndIf	All	<p>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.</p> <p>The better illustrations of IF - ELSE - ENDIF statements are included in Macro Examples herein, but, here are some simple illustrations:</p> <pre>If(var=1) Go(LabelX) EndIf</pre> <pre>If(var=1) Go(LabelX) Else Go(LabelY) EndIf</pre> <p>Also, see Else and Indentation practices in Chapter 5.</p>
IfPlatform() - ElseIfPlatform - EndIfPlatform	All	<p>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.</p> <p>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.</p> <p>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.</p> <p>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#_VPI D_11_273 and http://www.jdan.com/perfectscript/macros/appa_2.htm#_VPI D_AA_8057.</p>
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	<p>Indents a paragraph from the left margin.</p> <p>Indents both sides of a paragraph equally from left and right margins.</p>
Integer()	All	<p>This math command only works with "numeric" values. The value returned is the "whole number" of the identified variable or value.</p> <p>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.</p> <p>See Chapter 6, Working With Integers.</p>

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		<p>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.</p> <p>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.</p> <p>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:</p> <pre> 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 </pre>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>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:"OKBtn"; Left:67; Top:89; Width:85; Height:12; Style:OKBtn!; 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</p> <p>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:"CancelBtn"; Left:3; Top:3; Width:0; Height:0; Style:CancelBtn!; ButtonText:"Cancel") DialogAddText (Dialog:"vWait"; Control:"Static2"; Left:4; Top:38; Width:127; Height:17; Style:ShadowBox!+Center!; Text:"") Return</p> <p>Label(WaitMsg) DialogShow("vWait";"WordPerfect";cbWait) Return</p> <p>Label(cbWait) If(cbWait[3]="CancelBtn") Assert(CancelCondition!) EndIf Return</p> <p>Label(KillMsg) DialogDismiss("vWait";2) DialogDestroy("vWait") Return</p>
MacroDialogResult	All	<p>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.</p> <p>If it is desired to save the MacroDialogResult value, do so by var=MacroDialogResult.</p> <p>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.</p> <p>Most importantly, see the substantial discussion about this and related matters in Chapter 8, and the discussion in DialogDismiss(), above.</p> <p>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.:</p> <p>If(MacroDialogResult=1) [do this] Else [do that] EndIf</p> <p>If used without a callback routine, possible values assigned to MacroDialogResult will be:</p> <p>If an OKBtn closes the dialog, MacroDialogResult = 1;</p> <p>If a CancelBtn closes the dialog, MacroDialogResult =2;</p> <p>If a user-defined Button closes the dialog, MacroDialogResult = the RegionName (a case sensitive string, in quotes (e.g., "TipsBtn", "BackBtn"), exactly as you have defined the button's name in the dialog.</p> <p>Usage: var=MacroDialogResult, e.g., x=MacroDialogResult.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
MacroFilePlay()	All	<p>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.</p> <p>J. Dan Broadhead advises:</p> <p>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.</p> <p>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.</p> <p>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.</p>
MacroInfo()	All	<p>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:</p> <p>x=MacroInfo(ComputerName!) [string] x=MacroInfo(UserName!) [string] x=MacroInfo(PlatformVersion!) [string name, abbreviated] x=MacroInfo(PlatformVersionString!) [full string name]</p> <p>Also, see VersionInfo.</p>
MacrolsCompiled()	Wp8 & higher	<p>Determines whether a named macro is compiled or not. If so, a boolean value of <i>True</i> is returned; if not, <i>False</i>.</p> <p>Syntax: x=MacrolsCompiled("C:\MyMacs\test.wcm") or if the macro file name has been assigned to a variable, x=MacrolsCompiled(vMac). See MacroCompile, above, for an example.</p>
MacroPause	All	<p>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.</p>
MarginBottom(), MarginLeft(), MarginRight() and MarginTop()	All	<p>Sets the respective margins in inches.</p> <p>Example: MarginTop(0.75")</p>
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	<p>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.</p> <p>Additionally, consider J. Dan Broadhead's comments:</p> <p>The problem in the MessageBox getting 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).</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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:</p> <p>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:</p> <p>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:</p> <p>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".</p> <p>Self-explanatory closing enumerations are:</p> <p>OkCancel! YesNo! YesNoCancel! AbortRetryIgnore! RetryCancel!</p> <p>Values returned to the variable for the possible buttons are:</p> <p>OK=1 Cancel=2 Abort=3 Retry=4 Ignore=5 Yes=6 No=7</p> <p>Or, instead of testing for the above numeric values, you can do as J. Dan Broadhead recommends:</p> <p>I would encourage the use of the enumeration constants that this command returns, that can ease in the understanding of the macro code.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>As with all enumerations returned from commands, the enumerations must be prefixed by the name of the command they are returned from.</p> <p>So instead of checking for the value 6 to mean Yes, check for <code>MessageBox.YesButton!</code>:</p> <pre>x = MessageBox (....) If (x = MessageBox.YesButton!) [do this] EndIf</pre> <p>This will work back to at least version 7, and possibly version 6 as well.</p> <p>You can also combine the above enumerations with icon enumerations, if you want. Separate multiple enumerations by a "pipe" character:</p> <p><code>IconAsterisk! IconExclamation! IconHand! IconQuestion!</code></p> <p>So, with an icon enumeration, the command might be: <code>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!)</code></p> <p>Other variations are possible. See <code>MessageBox</code> in Macro help. And, see NTOC(), below, for adding hard returns in your message lines, and SetDefaultParent, generally.</p>
<code>Nest()</code>	All	Use Run() , instead. Used to run a macro, included for DOS compatibility.
<code>NTOC()</code>	All	<p>While other Number To Character (NTOC) commands exist, without doubt the most common is <code>NTOC(OF90Ah)</code>, a "hard return". So, if you define a variable, e.g., "H", as <code>H=NTOC(OF90Ah)</code>, you can use the variable H to insert a hard return, such as in a variable, e.g.:</p> <pre>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.</pre>
<code>NumStr()</code>	All	<p>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.</p> <p>So <code>var=NumStr(12345.5;0)</code> results in <code>var</code> equaling 123456, and <code>var=NumStr(12345.55;1)</code> SHOULD result in <code>var</code> 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 <code>NumStr()</code> 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 <code>NumStr()</code> commands in your WordPerfect macros.</p> <p>Usage: <code>var=NumStr(200.05)</code>, or <code>var=NumStr(var)</code>, where <code>var</code> is initially a numeric value, or <code>var=NumStr(200.05;0)</code> or <code>var=NumStr(var;16)</code>. See StrNum and Chapter 6, and, particularly, working with decimals.</p>
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 1 st 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 1 st 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 1 st 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.
PosCharPrevious	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	All	Moves the insertion point to the top of the document, after any initial codes.
PosDocVeryTop		Moves the insertion point to the very top of the document, before any initial codes.
PosLineBeg	All	Moves the insertion point to the beginning of the current line, after codes.
PosLineVeryBeg		Moves the insertion point to the beginning of the current line, before codes.
PosLineDown	All	Moves the insertion point down one line.
PosLineEnd	All	Moves the insertion point to the end of the current line, before any ending codes.
PosLineVeryEnd		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 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	<p>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.</p> <p>Usage: RegionAddListItem("RegionName"; "item"), e.g., RegionAddListItem("Test.B1"; "bananas") or RegionAddListItem("Test.B1"; "bananas"; "apples"; "pears").</p> <p>See Chapter 8 for more detail.</p>
RegionGet ...() commands, generally	All	<p>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.</p> <p>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.</p> <p>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 an 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.</p>
RegionGetCheck()	All	<p>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.</p> <p>Wp6.1 syntax: RegionGetCheck(var; "RegionName"), e.g., RegionGetCheck(var; "Test.B1")</p> <p>Wp7 & higher preferred syntax: var=RegionGetCheck("RegionName"), e.g., var=RegionGetCheck("Test.B1"), but note the following.</p> <p>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.</p>
RegionGetSelectedText()	All	<p>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.</p> <p>Wp6.1 syntax: RegionGetSelectedText(var; "RegionName"), e.g., RegionGetSelectedText(var; "Test.B1")</p> <p>Wp7 & higher preferred syntax: var=RegionGetSelectedText("RegionName"), e.g., var=RegionGetSelectedText("Test.B1"), but note the following.</p> <p>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.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
RegionGetWindowText()	All	<p>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.</p> <p>Wp6.1 syntax: RegionGetWindowText(var; "RegionName"), e.g., RegionGetWindowText(var; "Test.B1")</p> <p>Wp7 & higher preferred syntax: var=RegionGetWindowText("RegionName"), e.g., var=RegionGetWindowText("Test.B1"), but note the following.</p> <p>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.</p>
RegionRemoveListItem()	All	<p>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.</p> <p>Usage: RegionRemoveListItem("RegionName"; "item"), e.g., RegionRemoveListItem("Test.B1"; "bananas"). See Chapter 8 for more detail.</p>
RegionResetList()	All	<p>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.</p> <p>Usage: RegionResetList("RegionName"), e.g., RegionResetList("Test.B1"). See Chapter 8 for more detail.</p>
RegionSelectListItem()	All	<p>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.</p> <p>Usage: RegionSelectListItem("RegionName"; "item") e.g., RegionSelectListItem("Test.B1"; "No"). See Chapter 8 for more detail.</p>
RegionSetCheck()	All	<p>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.</p> <p>Usage: RegionSetCheck("RegionName"; numeric) e.g., RegionSetCheck("Test.B1"; Unchecked!) or RegionSetCheck("Test.B1"; Checked!). See Chapter 8 for more detail.</p>
RegionSetFocus()	All	<p>In a callback routine, this sets the active input position within a dialog. See Region Name Syntax, above.</p> <p>Usage: RegionSetFocus("RegionName"), e.g., RegionSetFocus("Test.B1"). See Chapter 8 for more detail.</p>
RegionSetWindowText()	All	<p>In a callback routine, this sets the text of a control within a dialog. See Region Name Syntax, above.</p> <p>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.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
RegionShowWindow()	All	<p>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.</p> <p>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:</p> <p style="padding-left: 40px;">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.</p> <p>Usage: RegionShowWindow("RegionName";enum), e.g., RegionShowWindow("Test.B1";Show!) or RegionShowWindow("Test.B1";Hide!).</p> <p>See Chapter 8 for more detail.</p>
Repeat (statement) Until(test)	All	<p>This is a loop command. The statement portion continuously executes until the test portion is the boolean value False.</p> <p>For example, this routine uses the command to strip commas out of a string:</p> <pre>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</pre> <p>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.</p>
RepeatValue(numeric)	All	<p>Sets the numeric value (number of times) the action which follows the command will be repeated.</p> <p>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).</p>
ReplaceAll()	All	<p>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.</p> <p>Example: ReplaceAll(Extended!) includes footnotes in the search and replace operation.</p>
ReplaceBackward()	All	<p>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.</p> <p>Example: ReplaceBackward(Extended!) will search backward through regular text, footnotes, endnotes, text boxes. ReplaceBackward will search backward through regular text.</p>

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
		<p>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").</p> <p>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.</p> <p>Also, see Chain().</p>
Search Commands	All	<p>These are links to various items used in writing search and/or replace routines, listed here for convenience.</p> <p>MatchExtendSelection MatchPositionAfter MatchPositionBefore MatchSelection ReplaceAll ReplaceBackward ReplaceCurrent ReplaceForward ReplaceString SearchCaseSensitive SearchNext SearchPrevious SearchString</p> <p>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.</p> <p>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.</p> <p>Examples:</p> <p>PosDocTop SearchString(var1) ReplaceString(var2) ReplaceForward SearchString(var1) MatchPositionAfter SearchNext</p> <p>SearchString("book") ReplaceString("pamphlet") MatchSelection SearchNext ReplaceCurrent</p> <p>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.</p>
SearchCaseSensitive()	All	<p>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.</p> <p>Usage: SearchCaseSensitive(Yes!)</p>
SearchNext()	All	<p>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.</p>
SearchPrevious()	All	<p>Same as SearchNext(), except that the search is backward.</p> <p>See Search Commands, above.</p>
SearchString()	All	<p>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.</p> <p>See Macro Toolbar Codes button in Chapter 5, and Search Commands, above, and search routines in ConvertFE.wcm.</p> <p>Examples: SearchString("bananas") and or SearchString("[Left Tab]"). In the latter, the Left Tab code was inserted using the Codes... button on the</p>

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	<p>It's best just to state J. Dan Broadhead's comments here:</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>Also, see MessageBox and, very broadly, the DialogShow reference in Chapter 5.</p>
SGML Commands	Wp8 & higher	None of the numerous SGML (Standard Government Markup Language) commands are included in this manual.
SingleSmartQuote()	All	<p>Turns on/off Single Smart Quotes with the On! or Off! parameter. See DoubleSmartQuote() and Quick... Commands, above.</p> <p>Usage: SingleSmartQuote(On!) or SingleSmartQuote(Off!)</p>
SingleSpaceInSentence	All	<p>Changes double spaces in a sentence to single spaces (or not) with the On! or Off! parameter.</p> <p>Usage: SingleSpaceInSentence(On!) or SingleSpaceInSentence(Off!)</p>
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	<p>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.</p> <p>Examples: SpellAsYouGo(Off!) or var=SpellAsYouGo. In the latter, var will be assigned a boolean true or false value.</p>
Style Commands	All	Except for the item below, none of the several Style commands are included in this manual.
StyleSystemOn()	All	<p>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.</p> <p>Example: StyleSystemOn(Doug) [no quotation marks]</p>
Str... Commands	Wp7 & higher	<p>Several Str... commands are covered in this manual, but others are not.</p> <p>Covered: StrFill; StrInsert; StrLen; StrNum; StrPos; StrReverse; StrToChars; StrTransform; StrTrim; SubStr</p> <p>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)</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
StrFill()	Wp7 & higher	<p>Replicates a string a specified number of times.</p> <p>Usage: vString=(n;"string"), e.g., <code>x=StrFill(3;"Doug")</code> <code>MessageBox("x";x)</code> //x="DougDougDoug"</p> <p>The 2nd parameter is optional. If not included, a "space" would be replicated.</p>
StrInsert()	Wp7 & higher	<p>Inserts characters into a string, replaces characters in a string, or removes characters in a string.</p> <p>Usage: vString=StrInsert(string; substring [optional]; begin at [n] [optional]; # of characters [n] [optional]).</p> <p>Paraphrasing WordPerfect's macro help:</p> <p>1st parameter: string (the original string)</p> <p>2nd parameter: substring: (optional – the string to insert; if missing, characters are removed from String)</p> <p>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)</p> <p>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)</p> <p>Examples:</p> <pre> 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" </pre>
StrLen(string)	All	<p>Returns the numeric length of characters in a value (numeric or string) or a variable, including spaces, e.g., <code>x=StrLen("Oklahoma")</code>, x will = 8; <code>x=StrLen(var)</code>, x will = the total characters, including spaces, in the "var" variable. See numerous macro examples in this paper, particularly Chapter 6, working with decimals.</p>
StrNum()	All	<p>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: <code>num=StrNum("abc20")</code>, <code>num=StrNum("-abc20")</code>, <code>num=StrNum(".abc20")</code>. And, while not producing error, exemplary results mixing numbers and characters are as follows: <code>num=StrNum("20abc30")</code> results in num equaling 20; <code>num=StrNum("200,000")</code> results in num equaling 200; <code>num=StrNum("200.005.008")</code> results in num equaling 200.005</p> <p>Usage: <code>num=StrNum("105.5")</code> or <code>num=StrNum(var)</code>, where the variable contains a numeric value.</p> <p>See NumStr and Chapter 6, working with decimals.</p>
StrPos()	All	<p>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.</p> <p>Usage: <code>num=StrPos(string [or var];substring)</code>, e.g., <code>x=StrPos(var;"p")</code> – if var is "oppressed", x will = 2 but if var is "kindly", x will = 0.</p> <p>See numerous macro examples in this paper, particularly Chapter 6, working with decimals and Automatic Data Conversion.</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
StrReverse()	Wp7 & later	<p>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.</p> <pre>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"</pre>
StrToChars()	Wp7 & later	<p>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.</p> <p>Usage: x=StrToChars(vString; enum; Str or enum)</p> <p>Example: x=StrToChars("12345.406.505";Keep!;".") //x=".." since only decimals have been kept</p> <p>Example: x=StrToChars("0.05%";Remove!;"%") //x="0.05" since "%" has been removed.</p> <p>Example: x=StrToChars(" 123,456.789";Remove!;Punctuation! WhiteSpace!) //x="123456789"</p> <p>See an example in Math.wcm.</p>
StrTransform()	Wp7 & later	<p>Converts a string of characters into other characters.</p> <p>Syntax: x=StrTransform(String: string; FromChars: string; [ToChars: string]; [Options: enumeration])</p> <p>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).</p> <p>Example: var=StrTransform(var;"%";"") or var=StrTransform(var;"%") //both remove all "%" signs from var</p> <p>Example: var=StrTransform(var;"",";","") //replaces all commas with semi-colons</p> <p>Example: If var="50.78 %",</p> <pre>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"</pre> <p>See examples in Math.wcm: Example 1; Example 2</p>
StrTrim()	Wp7 & Later	<p>Removes characters from a string.</p> <p>Syntax: x=StrTrim(String: string; [Length: numeric]; [Option: enumeration]; TrimChars: string or enumeration)</p> <p>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 parameter [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: Alphabetic!; AlphaNumeric! Numeric!; Punctuation!;</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>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 <i>if it is the only character identified in the 4th parameter</i> or by using the Punctuation!</p> <p>Example: var=StrTrim(var;;TrimLeft!;"0") // would eliminate all leading zeros from var, if any are present</p> <p>Example: If var="%50.78%",</p> <pre>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"</pre> <p>See example in Math.wcm: Example</p>
SubStr()	All	<p>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.</p> <p>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".</p> <p>See numerous macro examples in this paper, particularly Chapter 6, working with decimals and Automatic Data Conversion.</p>
SubstructureExit()	All	<p>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.</p> <p>Usage: SubstructureExit or SubstructureExit(Next!)</p>
Suppress()	All	<p>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).</p> <p>Example: Suppress(PageNumbering!) suppresses whatever page numbering you've got going on the current page.</p>
Switch(var) CaseOf EndSwitch	All	<p>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.</p> <p>Usage:</p> <pre>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</pre> <p>Example, where var="blue":</p> <pre>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</pre>

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>(Begin)</p> <p>CaseOf "orange"; "purple": MessageBox("The color is "+var;"")</p> <p>Go(vNext3)</p> <p>Default: MessageBox("No valid color was selected";"Start over")</p> <p>Go(Begin)</p> <p>EndSwitch</p> <p>EndIf</p> <p>Notice that it is possible to use multiple values in CaseOf, separated by semi-colons, as for "orange" and "purple", above.</p> <p>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.</p> <p>Note: while I tend to avoid the use of the MessageBox command, it was used above for simplicity in this description.</p>
SwitchDoc(num)	All	<p>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.</p> <p>Usage: SwitchDoc(number), either a number or a variable containing a numeric value of 1 ~ 9.</p>
Tab Commands	All	Several Tab commands other than the few shown below are available but they are not covered in this manual.
Tab	All	<p>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.</p> <p>Example: Tab Type("1.") Tab Type("My dog is good.")</p>
TabLeft()	All	<p>Inserts a hart left tab at the insertion point. Use the DotLeader! parameter if you want the tab preceded by a dot leader.</p> <p>Usage: TabLeft or TabLeft(DotLeader!) or TabLeft(Normal!)</p>
TabDecimal()	All	<p>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).</p> <p>HardReturn</p> <p>Tab Type("Books") DecimalTab DecimalTab Type("\$350.00") HardReturn</p> <p>Tab Type("Clothing") DecimalTab DecimalTab Type("75.00")</p> <p>HardReturn</p>

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!) AttributeAppearanceToggle(Underline!)

Command/Routine	Wp Ver	Syntax/Usage and Examples
		<p>Type("My dog has fleas.") - Text, when typed, will appear as <u>My dog has fleas.</u></p> <p>UnderlineSpaces(On!) AttributeAppearanceToggle(Underline!) Type("My dog has fleas.") - Text, when typed, will appear as <u>My dog has fleas.</u></p>
UnderlineTabs()	All	<p>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.</p> <p>Example: UnderlineTabs(On!) AttributeAppearanceToggle(Underline!) Tab Type("My dog has fleas.") - Text, when typed, will appear as <u>My dog has fleas.</u>, where a Tab precedes "My dog ..." and UnderlineSpaces is On.</p> <p>UnderlineTabs(Off!) AttributeAppearanceToggle(Underline!) Tab Type("My dog has fleas.") - Text, when typed, will appear as <u>My dog has fleas.</u>, where a Tab precedes "My dog ..." and UnderlineSpaces is On.</p>
VarErrChk()	All	<p>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().</p>
Variables, User Defined	All	<p>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.</p> <p>Note that this discussion has to do with user-defined variables, not the System Variables, above, 2, and not the special variable MacroDialogResult, above.</p> <p>The variable's name</p> <p>Duration of a variable</p> <p>Local variables</p> <p>Global variables</p> <p>Persist variables</p> <p>PersistAll</p> <p>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.</p> <p>Duration of A Variable: Variables can be Local, Global or Persist. While a variable exists, it uses memory.</p> <p>"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().</p> <p>"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().</p> <p>"Persist" variables stay in memory until discarded or until the WordPerfect session ends. As a practical matter, this will seldom be used.</p> <p>It is also possible to construct macros which retain variables in memory using the PersistAll command. That command is not developed in this manual.</p> <p>See Declare, above, for creating local variables with multiple elements,</p>

Command/Routine	Wp Ver	Syntax/Usage and Examples
Arrays		used in making Arrays.
VersionInfo()	Wp8 & higher	<p>Syntax: var=VersionInfo()</p> <p>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.</p> <p>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.</p> <p>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.</p>
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	<p>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.</p> <p>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.</p>
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.

NOTES

NOTES

NOTES

NOTES

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
? + < = A B C D E F G H I J K L M N O P Q R S T U V W X Y Z				
512 character line limit		Application		
!=		Arrays		Chapter 7 (Date Routines)
?DateDay		Assert		Chapter 8 (DialogShow, Callbacks)
?DateMonth		Attribute commands		Chapter 9 (Macro Examples)
?DateWeekDay		AttributeAppearanceOff		Chapter 10 (Glossary)
?DateYear		AttributeAppearanceOn		Check box
?DisplayMode		AttributeAppearanceToggle		CloseNoSave
?DocBlank		AttributeNormal		Closing & saving in Dialog Editor
?Endnote		AttributePosition		Column commands
?Font		AttributePositionToggle		Combo box
?FontSize		AttributeRelativeSize		Command Browser
?Footnote		AttributeRelativeSizeToggle		Commands button
?InTable		Automatic Data Conversion		Comment commands
?Justification		Average		Comments
?LeftChar		Beep		Compilation errors & Warnings
?LeftCode		Bitmap control		Compiling a macro
?LineSpacing		BlockProtect		Concatenation
?MajorVersion		Boolean value defined		Contents
?Name		Bookmark commands		Control Names
?NumberOpenDocuments		BookmarkCreate		Controls, Adding
?Path		BookmarkDelete		ConvertFE.wcm
?Path... commands		BookmarkFind		Copy
?PathCurrent		Books & Manuals		Copying a dialog to text
?PathDocument		Box commands		Ctrl+F10 starts and stops macro recording
?PathMacros		Call		Date system variables
?PathMacrosSupplemental		Callback routines		Date box
?PathSpreadsheet		Callback basics		Date commands
?QuickCorrect		Callbacks		Date commands not covered
?RightChar		Callbacks, Definition & Elements		DateCode
?RightCode		Callbacks, 1 at time		DateText
?Row		Callbacks, Region Names		DateFormat
?SelectedText		Callbacks, Why?		Date routines
?UnderlineSpaces		Callback Examples		DDE commands
?UnderlineTabs		Example 1		Declare
?Zoom		Example 2		Default button
+		Example 3		Default macro directory
-		Example 4		Delete commands
*		CallbackWait CallbackResume		DeleteCharNext
/		Cancel Button (Dialog Editor)		DeleteCharPrevious
//		Case manipulation		DeleteToEndOfLine
<		CaseOf		DeleteWord
<=		Center		DialogAdd... commands
<>		Chain		Dialog controls
=		Changing the default macro directory		DialogDefine
>		Chapter 1 (Other resources)		DialogDestroy
>=		Chapter 2 (Understanding macros)		DialogDismiss
A dumb keyboard macro example		Chapter 3 (First Things About Writing Macros)		DialogDestroy & Dialog Dismiss w/DialogShow
About the := sign		Chapter 4 (Controlling Flow)		DialogDisplay
Additional Resources		Chapter 5 (Dialog Editor; Writing Macros During Editing)		Dialog Editor
Adobe Acrobat Reader, Using		Chapter 6 (Math Routines)		accessing
Alignment of a dialog's controls				adding controls
All The Rest				aligning controls
AND				
Appearance attributes				

- associated variable
- Bitmap control
- Check box
- Closing & saving
- Combo box
- Controls
- Control size and placement
- Date box
- Default button
- Edit box
- Font
- Initial focus
- List box
- Ordering controls
- Push button
- Radio button
- Region, ...Control names
- Sequence/order
- Static Text box
- Viewer box
- Dialog font
- Dialog properties
- DialogShow, Beginning
 - Commands
- DialogShow/Callbacks
- DialogShow, Generally
 - Syntax
 - DialogShow w/o a Callback
 - Use of DialogDismiss & DialogDestroy
- DialogShow During Callbacks
- DialogShow w/CallbackWait
- DialogShow w/o CallbackWait
- Dialog Unit
- DirectoryCreate
- DirectoryExists
- Discard
- Display
- DisplayMode
- DisplayZoom
- DocInitialFont
- Don't use "" for numeric, boolean, etc.
- Double-Click to End Callback
- DoubleSmartQuote
- DropCap commands
- EditCopy
- EditCut
- Edit box
- Editing a macro
- Editing environment
- EditPaste
- EditPasteSimple
- Else
- End...commands
- EndNote commands
- Errors during macro compilation
- Exists
- Expressions
- FileExists
- FileInsert
- FileNew
- FileOpen
- Find or Find/Replace commands
- First Things
- FloatingCell commands
- Floating Cell problem
- FlushRight
- Font
- FontSize
- Footnote commands
- Forcing macro compilation
- Function EndFunc
- General definition of a macro
- General Notes
- General Notes About Table
- GetNumber
- GetString
- Global variables
- Go
- Graphics commands
- GraphicsLineCreate
- HardPageBreak
- HardReturn
- HardSpace
- Header commands
- Help Button (Dialog Editor)
- Help commands
- HLineCreate
- How macros work
- Hypertext commands
- Hyphen
- If - Else - EndIf
- IfPlatform - ElseIfPlatform - EndIfPlatform
 - Example (Chapter 8)
- Import commands
- Indent
- Indentation practices
- Index
- Initial focus in a dialog
- Internet macro resources
- Internet Q & A macro help
- Justification
- Labels commands
- Limitations of the Macro Help file
- Line commands
- Line limit of 512 characters
- LineSpacing
- List box
- List commands
- Local variables
- Macro commands, routines, included
- MacroDialogResult
- Macro directory
- MacroFilePlay
- MacroCompile
- MacroInfo
- MacroIsCompiled
- MacroPause
- Macro Shell, Making
- Macro Toolbar
- Macro Toolbar at a Glance
- Commands... button
- Codes button
- "Little" buttons
- Dialog Editor button
- Save & Compile button
- Using Stop & Record buttons
- Margin commands
- MatchExtendSelection
- MatchPositionAfter
- MatchPositionBefore
- MatchSelection
- Math and Comparison characters
- Math, Floating Cell Problem
- Math routines
- Merge commands
- MessageBox
- Naming the macro
- Nest
- NTOC
- OK Button (Dialog Editor)
- OLE commands
- OnCancel
- OnCancel Call
- OnError
- OnError Call
- OnNotFound
- OnNotFound Call
- Opening a dialog in Macro Editor
- Opening a macro file
- Operator precedence
- OR
- Ordering a dialog's controls
- Other Button (Dialog Editor)
- Outline commands
- Page Number commands
- PageNumber
- Paragraph commands
- ParagraphSpacing
- Parameters, generally
- PauseKey
- Persist variables
- PersistAll
- Playing Macros
- PosColumn commands
- PosCharacter
- PosCharNext
- PosCharPrevious
- PosDocBottom
- PosDocTop
- PosDocVeryTop
- Position attributes
- Position commands
- Position in table commands
- PosLineBeg
- PosLineDown
- PosLineEnd
- PosLineUp
- PosLineVeryBeg
- PosLineVeryEnd
- PosTableBegin
- PosTableCellBottom
- PosTableCellDown

PosTableCellNext	Reserved Words	SpellAsYouGo
PosTableCellPrevious	Return	Static Text box
PosTableCellTop	RevealCodes	Storage location of macros
PosTableCellUp	Run	StrFill
PosTableEnd	Saving a macro file	StrFraction
PosTableRowBegin	Search commands	StrInsert
PosTableRowEnd	also, see ConvertFE.wcm	StrIsChar
PosWordNext	SearchCaseSensitive	StrLen
PosWordPrevious	SearchNext	StrLeft
Pref commands	SearchPrevious	StrPad
Preface	SearchString	StrParseList
Print	Select anomalies, Wp10 & Later	StrPos
Procedure EndProc	Select commands	StrReverse
Prompt EndPrompt	SelectAll	StrRight
ProofReadAsYouGoOff	SelectCell	StrScan
Push button	SelectCellDown	StrToChars
Quick... commands	SelectCellDownArrow	StrTransform
QuickCorrect	SelectCellLeft	StrTrim
QuickCorrectQuickBulletsQry	SelectCellRight	StrUnit
QuickCorrectQuickBulletsSet	SelectCellUp	Style commands
QuickCorrectQuickIndentQry	SelectCellUpArrow	StyleSystemOn
QuickCorrectQuickIndentSet	SelectCharNext	SubStr
Quit	SelectCharPrevious	SubstructureExit
Radio button	SelectColumnBottom	Suppress
Rationale for including or excluding stuff	SelectColumnNext	Switch - EndSwitch
Reduction	SelectColumnPrevious	SwitchDoc
Region Names	SelectColumnTop	System Variables
Region Name Syntax	SelectDelete	Tab
Region... commands	SelectDocBottom	Tab commands
RegionAddListItem	SelectDocTop	TabDecimal
In Chapter 8	SelectDocVeryTop	Table commands
RegionGet... commands	SelectLineBegin	TabLeft
In Chapter 8	SelectLineDown	TablePosNextTable
RegionGetCheck	SelectLineEnd	TablePosPreviousTable
In Chapter 8	SelectLineUp	Template commands
RegionGet... Sample Code	SelectLineVeryEnd	TOA and TOC commands
RegionGetSelectedText	SelectMode	ToInitialCaps
In Chapter 8	SelectPage	ToLower
RegionGetWindowText	SelectPageNext	Top of document
In Chapter 8	SelectPagePrevious	ToUpper
RegionRemoveListItem	SelectParagraph	Type
In Chapter 8	SelectParagraphNext	UnderlineSpaces
RegionResetList	SelectParagraphPrevious	UnderlineTabs
In Chapter 8	SelectSentence	Understanding macros
RegionSelectListItem	SelectSentenceNext	Use of "" for string values
In Chapter 8	SelectSentencePrevious	Use of ()
RegionSetCheck	SelectTable	Use of Upper/Lower Case
In Chapter 8	SelectTableColumn	Use of VAR in this document
RegionSetFocus	SelectTableColumnExtendLeft	Using shortcut names
In Chapter 8	SelectTableColumnExtendRight	Usual macro file location
RegionSetWindowText	SelectTableRow	VarErrChk
In Chapter 8	SelectWord	Variable =
RegionShowWindow	SelectWordNext	Variable's duration
In Chapter 8	SelectWordPrevious	Variable's name
Repeat Until	Sequence of dialog controls	Variables, user defined
RepeatValue	SetDefaultParent	Version Control (Chapter 8)
ReplaceAll	SGML commands	VersionInfo
ReplaceBackward	Shipping macros	Viewer box
ReplaceCurrent	SingleSmartQuote	VLineCreate
ReplaceForward	SingleSpaceInSentence	WaitMessage callback example
ReplaceString	Size attributes	Web commands
	Sort commands	While EndWhile

Windows 2000 and XP issue
 WordPerfect's macro help
 WordPerfect 10 & up select issue
 WordPerfect 11 issue
 wp9select.wcm

Writing macros during macro
 editing
 Writing macros from scratch
 Writing macros using the
 keyboard

Zoom commands
 Zoom
 ZoomToFullPage
 ZoomToMarginWidth
 ZoomToPageWidth

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

NOTES

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

Release Notes

The initial release date of "A Common Person's WordPerfect Manual" was March 28, 2004. Substantive changes made thereafter, if any (i.e., other than typos or formatting changes), are noted here. Click on a link to go to the area affected. Pagination changed substantially with the June 22, 2004, revision. "Notes" pages were added at the end of each chapter in part to allow for "notes" but mainly to provide space for later additions or modifications with less likelihood of affecting pagination.

Chapter	Area Affected	Notes	Version
Preface & Contents	Added link to wp9select.wcm	New macro in Chapter 9	6/22/2004
Chapter 1			
Chapter 2			
Chapter 3	Wp10 & later select text method	Added link to wp9select.wcm	6/22/2004
Chapter 4	Label()	Clarified	3/31/2004
Chapter 5	Rewrote as needed for revised ConvertFE.wcm illustrations		6/22/2004
Chapter 6	Integer() Rewrote chapter as needed for revised Math.wcm examples & various Str... commands	Stated the range of valid values	6/22/2004 6/22/2004
Chapter 7			
Chapter 8	Math.wcm examples	Substantially rewritten	6/22/2004
Chapter 9	Math.wcm ConvertFE.wcm Wp9select.wcm Adds comment, Wp9Select.wcm	Substantially rewritten – see Note Substantially rewritten – see Note Added new macro Shows combining of Platform ID's	6/22/2004 6/22/2004 6/22/2004 7/2/2004
Chapter 10	Parameters and use of () Length of Single Code Line *, /, +, and - variable= or variable := >= ?MajorVersion ?Path ?PathMacrosSupplemental DialogAdd... DialogDismiss DialogShow [using CallbackWait] Exists(var;enum) GetString() IfPlatform... Integer Label() Label()	Added comments Clarified Clarified Corrected example Clarified Corrected reference Clarified Clarified Clarified Qualified Clarified Corrected Persistent! Corrected paragraph order Dropped incorrect examples Stated range of acceptable values Clarified Clarified	3/30/2004 3/30/2004 3/30/2004 3/31/2004 3/30/2004 3/30/2004 3/30/2004 3/30/2004 3/30/2004 3/30/2004 3/30/2004 3/30/2004 3/30/2004 3/30/2004 6/22/2004 3/30/2004 3/31/2004

Chapter	Area Affected	Notes	Version
	MacroDialogResult	Clarified	3/30/2004
	MacroDialogResult	Clarified and corrected	3/31/2004
	StrFill	Added	6/22/2004
	StrInsert	Added	6/22/2004
	StrReverse	Added	6/22/2004
	StToChars	Added	6/22/2004
	StrTransform	Added	6/22/2004
	StrTrim	Added	6/22/2004
	ToInitialCaps	Clarified	3/30/2004
	Type	Clarified	3/30/2004
Index	Index revised to add new items and links		6/22/2004

Note: Changes in Math.wcm. Many changes were made to the pre-6/22/2004 macro, as noted by the "green" comments in the macro itself. Generally, improvements were made in the math element to limit user numbers to 15 digits; to deal with calculated results returned by the macro in scientific notation; and to add additional macro capabilities (inserting result into boxes and printing results in columns format in a document).

Note: Changes in ConvertFE.wcm. Several changes were made to the pre-6/22/2004 macro, as noted by the "green" comments in the macro itself. Additional user options were added and the "Watch" item in the earlier macro was removed.

It is wholly expected that revisions will inevitably be made to this manual, and, when they are, they will appear here, with links to the sections affected.

Critiques and suggestions for corrections to improvement of this manual are invited by anyone via e-mail to me at loudenbk@swbell.com.

– Doug Loudenback, Oklahoma City, Oklahoma, USA.

[Top of Manual](#)