Contents

Introduction

PRIME 6 is a collection of add-on utilities for Microsoft Word for Windows 6.0, and is distributed as shareware. The only difference between the unregistered and the registered versions is that registered users receive a diskette version that contains no nag screens. Thanks in advance for registering!

PRIME 6 was developed by PRIME Consulting Group, Inc., the firm that co-authored the Microsoft Education Services (MES, formerly Microsoft University) WordBasic development course. The same firm whose principals (Timothy-James Lee and Lee Hudspeth) served as Technical Editors, along with the indefatigable Scott Krueger, on Leonhard and Chen's *Hacker's Guide to Word for Windows*. The same firm whose principals co-authored *The Underground Guide to Excel 5.0 for Windows* (Addison-Wesley). The same firm that developed PRIME 4, a Word 2 / PackRat 4 / FAXit integration application. The same firm that developed and self-published *Hacker's Guide to PackRat 4*.

PRIME Consulting Group, Inc. is a member of the Microsoft Solution Provider program, and is both a certified Consulting Partner and Training Partner. The firm's staff are certified to teach the following MES courses:

- *Application Development Using Microsoft Word*
- *Introduction to Programming for Microsoft Windows Using Microsoft Visual Basic 3.0*
- *Programming in Microsoft Visual Basic 3.0*

In addition, the firm offers its own two-day course on OLE 2.0 and DDE entitled *Putting OLE 2.0, VBA, and DDE to Work*.

Registration

PRIME 6 is shareware. You give it a whirl and if you like it, we'd very much appreciate it if you'd buy it within 30 days of first starting to tinker with it. The benefits of registering include:

- No nag screens.
- The most recent versions of all PRIME 6's components.
- Free telephone support via a toll call.
- Additional free support on CompuServe.
- Deals on autographed books by the authors of PRIME 6.

Thanks in advance for registering!

For single-user license agreement details, see License Agreement.

For site licensees: if your organization uses more than one copy of PRIME 6 at a time, you should license as many copies of PRIME 6 as you have copies of Microsoft Word for Windows 6. It's that simple.

PRIME 6 is $39.95US plus $4.50US shipping and handling, $9.50US outside North America. Site licenses (more than ten users) are available at considerable savings.

You can register right now by calling 800-788-0787, or 314-965-5630. We take Mastercard, Visa, American Express, purchase orders, and try hard to ship within 24 hours. To register by mail, send a check (in U.S. dollars, please) to:

PRIME Consulting Group, Inc.
c/o Advanced Support Group, Inc.
11900 Grant Place
Des Peres, Missouri USA 63131

License Agreement

This License Agreement is entered into between PRIME Consulting Group, Inc. as Licensor and recipient of enclosed diskette and related materials as Licensee.

This is a legal agreement between Licensor and Licensee. If Licensee does not agree to the terms of this Agreement, Licensee shall promptly return the disk(s) and accompanying material to Licensor.

Licensor hereby grants to Licensee the right to install the PRIME 6 components − including but not limited to DLLs, macros, templates, and WLLs ("PRIME 6") − on any one (1) personal computer. If Licensor installs PRIME 6 on a network server, Licensor agrees that one and only one user shall have access to that specific installed copy of PRIME 6. Use of PRIME 6 by more than one user is a violation of the terms of this agreement.

Use of PRIME 6 covered by this license by Licensee constitutes acceptance by Licensee of the terms and conditions of this agreement and Licensee agrees to be bound by the terms of same.

For the purpose of this agreement both parties hereby agree that the word "use" means loading files containing PRIME 6 features (P6.DOT or any other related template, DLL, or WLL supplied with PRIME 6) into RAM as well as installation on a hard disk or other storage device. In no event shall the term "use" include the right to modify, update, publicize, publish, republish, copyright, sell or transfer either this license, or the rights granted by this license, or the technology thereunder in relation thereto.

This license is limited to Licensee, which may access PRIME 6 from a hard disk, or any other method Licensee chooses to utilize, so long as Licensee otherwise complies with the terms of this License Agreement.

Licensee agrees and acknowledges that PRIME 6 was specifically written for Microsoft Word for Windows version 6.0 through 6.0a and Microsoft Windows 3.1, and that PRIME 6 may or may not work with any other version of these applications and operating systems/environments, either earlier or later.

LICENSOR DISCLAIMS, AND LICENSEE AGREES AND ACKNOWLEDGES THE DISCLAIMER OF ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL LICENSOR, OR ANY PERSON, FIRM OR ENTITY RELATED OR AFFILIATED THEREWITH, BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PURPOSE, BUSINESS INTERRUPTION, BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS WHATSOEVER, ARISING OUT OF THE USE OF, OR THE INABILITY TO USE, THE SUBJECT OF THIS LICENSE.

Licensee acknowledges that it has no rights whatsoever to copyright, in any country, anything in relation to PRIME 6, and promises and represents that it shall not attempt to do so, the same being exclusively vested in Licensor.

In the event Licensee breaches any term of this License Agreement it shall pay to Licensor, in addition to damages, all costs of suit and/or appeal to enforce the terms of this License Agreement, including, but not limited to, reasonable attorney's fees.

Both parties agree that this Agreement represents the complete and final agreement and understanding between the parties and all other prior and/or contemporaneous understandings or agreements are merged herein and shall have no further force and/or effect, and, that this Agreement is an "Integrated" agreement, and shall be governed exclusively by the laws of the United States of America and the State of California.

Acknowledgments

PRIME 6 was conceived and developed by PRIME Consulting Group, Inc.

We'd like to thank Woody Leonhard for his boundless support and encouragement. Ad infinitum. Amen.

Kudos and back-slapping are certainly due to Jonathan Sachs for exquisite programming, design, and technical writing contributions to Fast Spell Checker, Macro Manager, ResetChar (his idea all the way), Window Manager, Zoomer, and PRIME's API.

Thanks to Brian Gray of Aries Multimedia for a superb job on the setup routine. (Aries Multimedia is located in Fresno, California, 209-233-3630, CompuServe 73750,253.)

PRIME 6 wouldn't be what it is without the input we received from our beta testers (any PRIME 6 errors, goofs, or general misbehaviors are of course credited solely to PRIME Consulting Group), listed here in alphabetical order ... Dick Apcar, Steven Bretherick, Don Buchanan, Paul Cameron, Vince Chen, Bryan de Silva, Peter Deegan, John Donaldson, Malcolm Eaton, Stephen Edgington, Michael Evans, Pilar Ferrera, Brant Freer, Guy Gallo, Steve Gerber, Arthur Hixon, Michael Hyatt, Laurence Johansen, Joel Kanter, Jim Kauffman, John Kois, Scott Krueger, Milton Lassiter, Woody Leonhard, Patrick Long, Tom Looker, Ken Mocabee, Ellen Nagler, Rob Perelli-Minetti, Terence Pritchard, Adam Rodman, J. Bradford Sears, Terry Sherb, Paul Stokes, Stephen Stuart, Karl Swensson, Bill Walton, Brett Weiss, Eileen Wharmby, and Bob Yens. (If we left you out or misspelled your name, sorry, let us know and we'll send you some home-baked fudge, OK?)

Thanks to Tailor-Made Marketing and Advertising Programs for marketing and public relations consulting services par excellence. (Tailor-Made is located in Hermosa Beach, California, 310-318-7099).

We used WexTech Systems' amazing Doc-To-Help 1.5 to produce this *User's Guide* and help file for PRIME 6.

Related Topics:

Buy WOPR 6 Now (We're Not Kidding)!

Buy WOPR 6 Now (We're Not Kidding)!

Call Advanced Support Group at 800-659-4696 or 314-965-5630.

A synopsis of WOPR 6's features:

- Enveloper
- File Manager
- Stellar Spellar
- Toolbar Editor
- File New
- 2x4
- TSS Address Book Lite
- Clip Art Scrapbook
- WinBreak Lite
- and a dozen or so LittleWOPRs

Minimum Requirements

To successfully run PRIME 6 you need the following.

- Microsoft Windows 3.1 or later
- Microsoft Word for Windows 6.0 or later (to find out about or order maintenance releases of Word call Microsoft at 800-426-9400)

How to Install PRIME 6

PRIME 6 is installed via an automated setup routine (SETUP.EXE).

Follow these steps to add PRIME 6 to your system. (If you're *not* installing from a floppy, but instead copied the floppy's contents onto your hard disk, replace "a:\" throughout these instructions with the appropriate storage path on your hard disk.)

1. Exit Word 6 if it's already running.
2. Go to Program Manager, select File then choose Run.
3. Type "a:\setup.exe" (without the quotes) and press Enter or click OK. (If you're installing from your b: diskette drive, type "b:\setup.exe".)
4. Follow the instructions on the installation screen.
5. When installation is complete, it leaves you in Notepad to review the latest PRIME 6 README.TXT file. Exit Notepad when you've reviewed any relevant late-breaking news.
6. Start Word 6.

   You'll see PRIME 6's toolbar displayed. If for some reason you don't see it, select View on the main menu bar, select Toolbars, scroll down the Toolbars list until you see "PRIME 6.0," select the PRIME 6.0 check box, then click OK.
7. Open P6INST.DOC which can be found on your installation diskette (or hard disk if you copied the floppy's contents there).
8. Double-click the installation button to copy ResetChar to NORMAL.DOT. NORMAL.DOT is updated and saved automatically for you.
9. Close P6INST.DOC.

   Installation of PRIME 6 is now complete. Enjoy.

Each PRIME 6 utility has its own menu item hanging off various standard menu bar commands, as well as its own button on the PRIME 6 toolbar.

For instructions on how to uninstall PRIME 6, see How to Uninstall PRIME 6.

File Inventory

A file inventory follows. Each file's required storage path is shown in parentheses.

- CSLIDER.DLL (Word's Programdir) – custom control used by P6ZOOM.WLL
- CTL3D.DLL (Windows System) – substitutes 3D controls for Windows' standard "flat" controls
- MMASTER.INI (Windows) – settings preferences file used by various utilities; PRIME 6 creates this file for you
- P6.DOT (Word's Startup-Path) – global add-in template where all the utilities live
- P6.HLP (Windows) – help file
- P6FAST.WLL (Word's Tools-Path) – Word add-in library used by PRIME Fast Spell Checker
- P6INST.DOC (master diskette) – non-global template used once to install customized ResetChar to NORMAL.DOT
- P6ZOOM.WLL (Word's Startup-Path) – Word add-in library used by PRIME Zoomer
- README.TXT (master diskette) – general last-minute comments and stuff

Technical Support

For technical support call 314-965-5630, fax 314-966-1833, or use CompuServe (type GO WOPR).

How to Uninstall PRIME 6

Follow these steps to remove PRIME 6 from your system.

1. Start Word 6 if it isn't already running, and if it is already running, close all open documents.
2. Open P6INST.DOC which can be found on your installation diskette (or hard disk if you copied the floppy's contents there).
3. Double-click the uninstallation button and answer all the questions as appropriate.

   When the uninstallation is complete, remember that you have to manually delete PRIME 6's version of ResetChar from NORMAL.DOT. To do this, follow these steps.

   a. Select Tools from the menu bar.

   b. Choose Macro from the menu pull-down list.

   c. Choose Normal.dot (Global Template) from the **Macros Available In** drop-down box.

   d. Select ResetChar in the Macro Name list.

   e. Click the Delete button and answer Yes to the question "Do you want to delete macro ResetChar?"

   f. Click the Close button.
4. File Exit and restart Word.

   Tools Options Save contains a **Prompt to Save Normal.dot** check box. If it's selected on your system then when Word exits it detects that NORMAL.DOT has changed and explicitly ask you if you want to save NORMAL.DOT. *Answer Yes*. However, if it's cleared on your system then when Word exits it saves NORMAL.DOT for you automatically.

AutoText Lister

Word's native Edit AutoText dialog lumps template and global AutoText entries into a single undifferentiated list. PRIMEAutoTextLister splits this list into two sets of controls so you can finally view these two groups of AutoText based on their context. This utility allows you to view template and global AutoText contents and statistics side-by-side, catalog them all into a table, and insert selected AutoText into the current document.

The two AutoText name list boxes (**Global Templates and Add-ins** and **Template**) display AutoText entry names in the order in which WordBasic's **AutoTextName$()** command locates them by context. *The Word Developer's Kit* and Word's on-line help are wrong. *The Word Developer's Kit* states on page 268, "AutoText entries are listed in alphabetic order." This is true for a non-global template. It is also true that Word's native EditAutoText dialog displays entry names in alphabetic order *across all global and add-in templates*. But if you build a global entry name list yourself using **CountAutoTextEntries()** and **AutoTextName$()** with the classic technique

```
Global = 0

For i = 0 to CountAutoTextEntries(Global) - 1

AutoTextGlobal$(i) = AutoTextName$(i + 1, Global)

Next i
```

then you get a list of entry names sorted *first in add-in load order* and *second by entry name*. This poses some significant problems in terms of accessing global entries with duplicate names, and AutoText Lister overcomes these problems.

For information about Word and WordBasic bugs related to AutoText, see Word and WordBasic Bugs and Workarounds.

The **Contents** frame displays the selected AutoText entry's unformatted contents. If the character count of the entry exceeds 255, only the first 255 characters are displayed in the frame. This is because WordBasic's label control has a maximum capacity of 255 characters. If the AutoText entry contains a picture *and no other characters*, that picture is properly displayed in the **Contents** frame.

The **Statistics** frame displays some statistical information about the selected AutoText entry.

- Character count
- Entry number of total number (in the case of global entries, the total number is equal to the sum total of all entries in NORMAL.DOT and any loaded global templates)

The **Insert** button inserts the current AutoText entry into the current document at the insertion point (where the cursor is).

The **Log All** button generates a four-column table in a scratch document (based on the current template) that you can then print or save. Unlike the **Contents** frame, which does not display any character formatting, the **Log All** table uses

```
EditAutoText .Name = AutoTextName$, .InsertAs = 0, .Insert
```

to insert the formatted contents of each entry. (.InsertAs = 0 counterintuitively represents insertion of an entry in its formatted form, and InsertAs = 1 inserts it as plain text. However, this is backward compatible with WordBasic 2.0 in which the EditGlossary parameter was .InsertAsText with no assigned value.) In the case of duplicate-name global AutoText entries, AutoText Lister jumps through the appropriate hoops to provide the formatted contents. Remember, an unadulterated call to EditAutoText referring to the second or subsequent duplicate-name entry would always incorrectly return the first duplicate-name entry, but AutoText Lister's logic works around this problem. Of course, lengthy text blocks or large pictures produce a table with one or more tall rows.

Related Topics:

Table Column Headings
AutoText Entries That Contain Tables
Duplicate AutoText Names

Table Column Headings

AutoText Lister's log table column headings follow.

- Counter
- Context: Name
- AutoText Contents
- AutoText Statistics

Entries containing more than 255 characters are fully displayed in the log table, even though they are truncated when displayed in the dialog box (as explained earlier).

AutoText Entries That Contain Tables

AutoText entries that contain tables can't be logged. This is because tables can't be inserted into tables. The log entry for such items reads "<NOTE: This AutoText entry cannot be logged because it contains a table.>"

Duplicate AutoText Names

For information on a bug related to duplicate-named global AutoText entries, see GetAutoText$() Always Returns Contents of First Duplicate-name Global AutoText.

Bookmark Manager

View, modify, catalog (in a table), goto, and delete bookmarks and their contents, and display bookmark statistics. This utility properly handles the recalcitrant Smart Cut and Paste feature of Word 6. Neither Word's Edit Bookmark nor Edit Go To dialogs display a bookmark's contents, PRIME Bookmark Manager does!

Bookmark Manager uses the custom subroutine **sRedefineBookmark()** to redefine a bookmark to text you type in the utility's dialog box. You can use this same subroutine yourself. For a discussion of this custom subroutine, and a detailed description of how Word misbehaves when you use WordBasic − *without* sRedefineBookmark() − to insert text into an existing bookmark, see PRIMELib.sRedefineBookmark(DestinationBkmk$, NewText$).

The **Bookmark Names** list box displays the current document's bookmarks sorted in alphabetical order.

The **Bookmark Contents** text box displays the selected bookmark's unformatted contents. If the character count of the bookmark exceeds 255, only the first 255 characters are displayed. This is because WordBasic's text box control has a maximum capacity of 255 characters.

The **Statistics** frame displays some statistical information about the selected bookmark.

- Character count
- Section number
- Page number
- Bookmark number of total number (*name order*)
- Bookmark number of total number (*location order*) − for information on deriving a bookmark's position see Form Field Bookmarks Don't Exist According to SelInfo(29)

The **Goto Now & Wait** button moves to and selects the current bookmark, and leaves the dialog box displayed.

The **Goto Now & Close** button moves to and selects the current bookmark, and closes the dialog box.

The **Goto Native** button invokes Word's Edit Go To dialog and pokes the selected bookmark name into the **Enter Bookmark Name** drop-down. However, it waits for you to click the native dialog's Go To button (or press Enter) to take the action. To exit the native dialog, click on Close or press Esc.

The **Delete** button deletes the selected bookmark. It warns you with a message box first. If you proceed with the deletion, the bookmark list is refreshed and the **Bookmark Names** list box reflects the deletion.

The **Log All** button generates a seven-column table in a scratch document (based on the current template) that you can then print or save. The log includes each bookmark in its entirety, including character formatting and graphics (if any), and all relevant bookmark statistics. If the bookmark includes one or more paragraph marks, those paragraphs' styles are preserved because the scratch log document is based on the current template.

The **Redefine** button redefines the selected bookmark to the contents of the **Bookmark Contents** text box. If the bookmark originally contained more than 255 characters, the utility warns that you are about to truncate it and you can cancel the operation or continue. The redefinition works properly regardless of your current **Typing Replaces Selection** and **Use Smart Cut and Paste** options (both located in the Tools Options Edit tab). For information on the custom subroutine that performs the redefinition, see PRIMELib.sRedefineBookmark(DestinationBkmk$, NewText$).

The **Toggle Bookmarks** button toggles the state of the Tools Options View Bookmarks setting, and leaves the dialog box displayed.

Related Topics:

Table Column Headings

Bookmarks That Contain Tables
Bookmarks and Protected Documents

Table Column Headings

Bookmark Manager's log table column headings follow.

- Name Order
- Location Order
- Chars
- Section
- Page
- Name
- Contents

Bookmarks containing more than 255 characters are fully displayed in the log table, even though they are truncated when displayed in the dialog box (as explained earlier).

Bookmarks That Contain Tables

Bookmarks that contain tables can't be logged. This is because tables can't be inserted into tables. The log entry for such items reads "<NOTE: This bookmark cannot be logged because it contains a table.>"

Bookmarks and Protected Documents

Word's bookmark features are not available for protected documents. Bookmark Manager warns you of this if the current document is protected, and exits gracefully. To unprotect a document, select Tools and choose Unprotect Document. Then you can run PRIME Bookmark Manager.

Create Program Manager Icon

From within Word, create a Program Manager Item for the current document in the Program Group of your choice. You can also add a new Program Group, all from the same dialog box.

When invoked, this utility first checks to see if the current document has been saved. If it has not been saved, then it prompts you via the Save As dialog to save the file. If you save the file, the utility's main dialog box appears, but if you cancel out of Save As then a message box explains you must save a file at least once before creating a Program Manager icon for it.

The **Choose Existing Group** list box displays all Program Groups sorted alphabetically.

The **Enter New Item Description** text box displays the current document name in its DOS 8.3 filename format. You can overtype this if you like. Program Item descriptions are limited to a maximum of 40 characters.

The **Items Currently in Chosen Group** list box displays all Program Items in the current Group, sorted alphabetically.

The **Add Group** button invokes a new dialog box panel with an **Add a New Group** text box and the prior **Enter New Item Description** text box. Once you enter a new Group and click OK, the new Group is added to Program Manager and the current document is added to the new Group. The new Item's description comes from the **Enter New Item Description** text box.

Document Variable Manager

Document variables are a cool new feature of Word 6.0, but there's no native feature for manipulating them. PRIME Document Variable Manager to the rescue! This utility allows you to add, view, modify, delete and catalog (in a table) document variables and their contents.

If there are no document variables in the current document, the utility displays a **New Variable Name** text box in which you enter the variable name, and an **Add Variable Contents** text box in which you enter the contents, up to the maximum of 255 characters allowed in a text box by WordBasic. Click the **Define** button to consummate the variable definition. The new variable now appears in the list.

If you want to add a document variable with more than 255 characters, you'll have to roll your own WordBasic code like this.

```
Count = 1024
HugeString$ = String$(Count, "*")       ' Count <= 65280
VariableName$ = "HugeVar"
If Not SetDocumentVar(VariableName$, HugeString$) Then
MsgBox "Could not add " + VariableName$
End If
```

If there is at least one document variable then the utility replaces the **New Variable Name** control with a **Variables in Current Document** list box that displays document variable names sorted alphabetically.

The **Add New** button adds a new document variable.

The **Change** button provides an edit mode in which you can replace the current variable contents with new information by typing into the **Edit Contents** text box. Click **Define** to affect the change, or **Close** to abort the change and quit the utility.

The **Log All** button generates a two-column table in a scratch document that you can then print or save.

Document variables containing more than 255 characters are fully displayed in the log table, even though they are truncated when displayed in the dialog box (as explained earlier).

For information about Word and WordBasic bugs related to document variables, see Word and WordBasic Bugs and Workarounds.

Related Topics:

Table Column Headings

Table Column Headings

Document Variable Manager's log table column headings follow.

- Variable Name
- Variable Contents

Fast Spell Checker

Tired of sipping cold coffee while waiting for Word's spell checker to move laboriously from one misspelled word to another ... with you shackled to the keyboard? Break the habit! Let PRIME Fast Spell Checker do all the work for you *by showing you all misspelled words at once*. That's right. Turn spell checking upside down and let your Personal Computer do some Computing! And get a complete word list with word frequencies to boot! Fast Spell Checker scans your document quickly then presents a single list of all misspelled words which you can correct as needed.

The section Jump Start -- Read This First, Please will get you started in a jiffy.

For details about Fast Spell Checker's main features see Making Corrections.

Related Topics:

Jump Start -- Read This First, Please
How Fast Spell Checker Uses Custom Dictionaries
Making Corrections
Miscellaneous Fast Spell Checker Issues

Jump Start -- Read This First, Please

To get started using Fast Spell Checker (FSC) in a jiffy, here's all you need to know. There are two different ways to approach the process of correcting a misspelled word. One way is to have FSC show you each occurrence of a misspelled word in its context within the document, one occurrence at a time. The other way is to *not* have FSC show you each occurrence of a misspelled word in its context, rather, to correct all occurrences of a misspelled word in one fell swoop, *right now*. The following two sections explore each of these approaches in turn.

Related Topics:

Show Me Each Occurrence of a Misspelled Word in Its Context Within the Document
Correct All Occurrences of a Misspelled Word in One Fell Swoop

Show Me Each Occurrence of a Misspelled Word in Its Context Within the Document

The following steps assume you want FSC to show you each occurrence of a misspelled word in its context within the document, one occurrence at a time.

1. Start Fast Spell Checker via the PRIME 6 toolbar button or the Tools menu drop-down.
2. Click **Suggest** for the first word in the **Not in Dictionary** list box. This makes Word suggest corrections and goes to the first occurrence of this word. (You may have to drag FSC's dialog box around on the screen to see the word in context. The word will be highlighted.)
3. If you don't like Word's first suggestion, click the suggestion you prefer, or type a correction into the **Change To** text box. If the suggestion now in the **Change To** text box does not match the case of the current instance of the word in the document, click **First** and FSC attempts to normalize the suggestion's case to match that of the current instance.

   If you want to correct this instance, click **Change**. FSC corrects the instance and highlights the next one.

   If you don't want to correct this instance, click **>>** to go to the next occurrence.
4. Repeat until you have reviewed every instance of the misspelled word.

   If you correct the last instance of the word, FSC automatically highlights the next misspelled word. If you don't correct the last instance of the word, when you click **>>** FSC displays a message saying there are no more instances. When you see this message, click **Ignore** to highlight the next misspelled word in the **Not in Dictionary** list.

   When you have corrected the last misspelled word, the **Not in Dictionary** list box is empty. If you want to see what your document's word list looks like, click **Wordlist**. (The speed with which the word list is produced depends on the word count of the document and your PC's configuration.) The utility shows you percent complete while it prepares the word list.
5. If you are done, click **Close**.

If none of the suggestions are right (or if there are no suggestions, or you didn't bother clicking **Suggest** at all), you can type a correction directly into the **Change To** text box. But note that if you do this without clicking **Suggest** first, FSC won't enable the **Change** and **Change All** buttons, because WordBasic won't notify it that you have entered a correction until the focus shifts to a control other than the **Change To** box.

Correct All Occurrences of a Misspelled Word in One Fell Swoop

The following steps assume you do *not* want FSC to show you each occurrence of a misspelled word in its context within the document, one occurrence at a time. Rather, you want FSC to correct all occurrences of a misspelled word in one fell swoop, *right now*.

1. Start Fast Spell Checker via the PRIME 6 toolbar button or the Tools menu drop-down.
2. Click **Suggest** for the first word in the **Not in Dictionary** list box. This makes Word suggest corrections and goes to the first occurrence of this word. (You may have to drag FSC's dialog box around on the screen to see the word in context. The word will be highlighted.)
3. If you don't like Word's first suggestion, click the suggestion you prefer, or type a correction into the **Change To** text box.

   To correct all instances of this word, click **Change All**.

   Be aware that the case of the text in the **Change To** text box is used throughout your document for all instances of this misspelled word, regardless of each instance's original capitalization. For example, you have both "chease" and "Chease" in your document. FSC locates the first instance of the misspelled word and you select "cheese" from the suggestions list and click **Change All**. FSC changes both misspelled instances to "cheese" (all lower case).

   If there's just one suggestion you don't have to click the combo box list item, just click **Change All**. (An example of a word that returns only one suggestion from the main dictionary is "afflictive.")
4. Repeat until you have corrected the last misspelled word.

   When you have corrected the last misspelled word, the **Not in Dictionary** list box is empty. If you want to see what your document's word list looks like, click **Wordlist**. (The speed with which the word list is produced depends on the word count of the document and your PC's configuration.) The utility shows you percent complete while it prepares the word list.
5. If you are done, click **Close**.

If none of the suggestions are right (or if there are no suggestions, or you didn't bother clicking **Suggest** at all), you can type a correction directly into the **Change To** text box. But note that if you do this without clicking **Suggest** first, FSC won't enable the **Change** and **Change All** buttons, because WordBasic won't notify it that you have entered a correction until the focus shifts to a control other than the **Change To** box.

How Fast Spell Checker Uses Custom Dictionaries

When Fast Spell Checker (FSC) calls Word's spelling checker to check an unrecognized word, it uses whatever custom dictionaries Word is currently configured to use. You control custom dictionaries manually using Word's Tools Options Spelling parameters. To examine these parameters, do the following.

1. Select Tools from the menu bar.
2. Select Options.
3. Click on the Spelling tab.
4. Look in the **Custom Dictionaries** frame for a list of active custom dictionaries.
5. Click **New**, **Edit**, **Add**, or **Remove** to manipulate your custom dictionaries as needed.
6. Click **OK** or **Cancel** to quit.

Making Corrections

When Fast Spell Checker (FSC) has finished scanning the document it displays its main dialog box with appropriate controls enabled.

The **Not in Dictionary** box lists the misspelled words that FSC found in the document. You can work with any word at any time by clicking it. When you click a word in the **Not in Dictionary** box, FSC copies it to the **Change To** box.

FSC's controls are described below.

- **First** button: Selects the first occurrence of this misspelled word in the document.
- **>>** button: Selects the next occurrence of this misspelled word in the document.
- **<<** button: Selects the previous occurrence of this misspelled word in the document.
- **Suggest** button: Uses Word's spelling checker to suggest corrections for the misspelled word and display them in the **Change To** combo box. Clicking **Suggest** *also selects the first occurrence of the word if no occurrence has been selected yet*.

  FSC displays the spelling checker's suggestions in the list box under **Change To**, and copies the first one into the **Change To** text box. If you don't like Word's first suggestion, click the suggestion you prefer, or type the correction into the **Change To** text box.
- **Change** button: Corrects the selected instance of the misspelled word by replacing it with the correction in the **Change To** text box. Note that the **Change** button is grayed out until you cause FSC to go to an occurrence in the document of the current misspelled word; then **Change** becomes an available control.

  After FSC corrects a word it automatically selects the next occurrence of that word. If there are no more occurrences of the word in the document it deletes the word from the **Not in Dictionary** list and highlights the next word in the list.
- **Change All** button: Corrects all instances of the misspelled word. Also deletes the word from the **Not in Dictionary** list and highlights the next word in the list.
- **Ignore** button: Tells FSC to not try to correct instances of this misspelled word. Also deletes the word from the **Not in Dictionary** list and highlights the next word in the list.

  Note that an ignored word will appear in the **Not in Dictionary** list the next time you run FSC on this document.
- **Add** button: FSC adds the current contents of the **Change To** text box to the first custom dictionary.

  "First" here means the first custom dictionary listed in your Tools Options Spelling filecard's Custom Dictionaries list box. For more information see How Fast Spell Checker Uses Custom Dictionaries. The addition to the custom dictionary is case sensitive, that is, the exact case of the **Change To** text box's contents is updated to the custom dictionary.
- **Accept** button: Treats this word as a correctly spelled word. Also deletes the word from the **Not in Dictionary** list and highlights the next word in the list.

  This control changes the word's status in the FSC's internal vocabulary to "valid, not in dictionary" (just another way of describing a misspelled word that has been "accepted") so that the word will not appear in the **Not in Dictionary** list in future runs of FSC.
- **Wordlist** button: Creates a word frequency list which is a list of all of the words that occur in the document, in alphabetic order.

  For each word, the list presents the word itself; the word's status in the document's vocabulary; the number of times the word occurs in the document; and the percentage of the document's total words that are this word. The word frequency list is stored as a table in a new document, which you can save and/or print. The table also includes the fully qualified document name and a date/time stamp.

- **Close** button: Closes the correction phase of FSC. You can click **Close** at any time; you do not have to wait until the **Not in Dictionary** list is empty.

Miscellaneous Fast Spell Checker Issues

Following is a collection of miscellaneous information about Fast Spell Checker.

Related Topics:

Duplicate Words
Headers, Footers, Footnotes, and Annotations
Insensitive to Certain Unusual Capitalizations
Ignores Punctuated Words Inside a Sentence
Producing an Updated Word Frequency List
Word Errors When Formatting Large Word Frequency Lists
Word Bug Affects FSC Scan When a Table Is at the Start of a Document
Cursor Starting Position, Document Variables, and a Dirty Document
Why Misspelled Words Are Shown in Uppercase in the "Not in Dictionary" List Box
Scan Speed Accelerates As FSC Progresses Through a Large Document
Location of Word's Main Dictionary
Version Compatibility

Duplicate Words

FSC does not currently detect duplicate words like this, "The lady or the the tiger."

We plan to enhance FSC to detect such cases in a future release.

Headers, Footers, Footnotes, and Annotations

FSC does not currently check spelling of text in header, footer, footnote, or annotation panes.

We plan to enhance FSC to check these panes in a future release.

Insensitive to Certain Unusual Capitalizations

If you have an unusual capitalization – like "LaserJet" – stored in a custom dictionary, FSC currently ignores any capitalization of this word in a document. For example, FSC currently does not think "laserjet" or "LAseRjeT" or "LASERJET" are misspelled when "LaserJet" is in a custom dictionary.

We plan to enhance FSC to be sensitive to these same-word capitalization variations in a future release.

Ignores Punctuated Words Inside a Sentence

FSC currently looks at a punctuated word like "Mr." in the sentence "Mr. Cheese goes to Hollywood." and thinks "Mr" is a misspelling. If you click **Suggest** you get back "Mr." and if you **Change** it, you end up with "Mr.. Cheese goes to Hollywood." (note the two periods now in "Mr.."). A similar problematic word is "etc." The workaround is to simply click **Ignore** for these punctuated words.

We plan to enhance FSC to ignore such cases in a future release.

Producing an Updated Word Frequency List

FSC currently produces a word frequency list based on the words it *originally* scanned in the current session, and ignores any changes you have made in the current session. The workaround is to finish your spelling corrections, **Close** FSC, run it again, and then produce a word list that is now fully refreshed and current.

We plan to enhance FSC in this regard in a future release.

Word Errors When Formatting Large Word Frequency Lists

If your document contains a very large number of unique words (the threshold value depends on your PC's configuration and current system resources), while FSC is formatting the word frequency list table you may encounter some Word error messages. Typically Word will report "Word has insufficient memory. You will not be able to undo this action once it is completed. Do you want to continue?" [Yes/No/Help] In many cases you can Click Yes and things will work fine from there. However, at some point the word frequency list table becomes so large that Word simply cannot perform the Convert Text to Table operation and fails with a WordBasic Err 102 "Command failed" message, followed by a "There are too many edits in the document. This operation will be incomplete. Save your work." message. Word then leaves you in the word frequency list document with all the words and frequency statistics properly organized but not formatted as a table. We suggest you save the document and format the large list as several smaller tables.

In addition to the above error messages you may see a WordBasic Err 7 "Out of memory" message. (In our testing, this Err 7 condition only occurred if we tried to run a word frequency list for a large word-count document immediately after receiving the messages described in the prior paragraph.) At this point you should exit Word, then exit Windows, then restart Windows and Word.

We plan to enhance FSC in this regard in a future release.

Word Bug Affects FSC Scan When a Table Is at the Start of a Document

Due to a bug in Word, if a document begins with a table, the scan text FSC gets back from Word while inside the table is comprised of a merger of each word in adjacent cell pairs. You can avoid this problem by simply inserting a single paragraph mark before the table, then the Word bug doesn't manifest itself. Then run FSC, make your corrections, and delete the leading paragraph.

If Microsoft fixes this bug in a post-6.0a maintenance release of Word then this issue goes away. Otherwise, we plan to resolve this unusual situation in a future release.

Cursor Starting Position, Document Variables, and a Dirty Document

Let's say you run FSC on a document that FSC has never scanned before, make no spelling corrections, and simply close FSC once it has built its list of misspelled words. If you then close the document, Word asks you "Do you want to save changes to ...?" That's because FSC created a throw-away bookmark to mark the cursor's starting position and destroyed it when it was done. FSC also added a document variable to the document (which is where FSC's vocabulary is stored). Both these actions, either separately or together, "dirty" a document, even though none of the document text actually changed. Your response to Word's "Do you want to save changes to ...?" query should be to click Yes and save changes to the document.

We plan to enhance FSC's intelligence in the case where no corrections are made in a future release.

Why Misspelled Words Are Shown in Uppercase in the "Not in Dictionary" List Box

FSC is based on the notion of a single list of all misspelled words (often called "batch spell checking") as opposed to Word's way of walking through a document one misspelled word at a time (often called "serial spell checking"). Since there can easily be more than one instance of a particular misspelled word, we had to make a decision to show each misspelled word in the list either in all lower case or all upper case. We chose upper case.

This is no way affects how FSC corrects a misspelled word, as you can see when you Suggest and then Change (or Change All) a word. You'll see that FSC preserves the capitalization of each instance of the word, exactly the same way Word does when you use its spell checker.

Scan Speed Accelerates As FSC Progresses Through a Large Document

FSC's scan speeds up as it progresses through a long document. This is apparent from the rate of change in the "percent complete" message. It happens because the WLL spell-checks only the first occurrence of each word in the text. The more text you process, the greater a proportion of words have already been checked, and the less work the spelling engine has to do.

Location of Word's Main Dictionary

P6FAST.WLL finds the main dictionary by referring to WINWORD6.INI. First it looks for an entry for 'Spelling 1033,0' (American English). If it can't find that it looks for 'Spelling 2057,0' (British); if it can't find that, 'Spelling 3081,0' (Australian). If it can't find that it displays the message box

PRIME Fast Spell Checker can't locate the main spelling dictionary! Check WINWORD6.INI.

Then it quits.

If it finds the dictionary in WINWORD6.INI but can't open it, you get the "CSAPI initialization error!" message box with a major code of 2 and a minor code that depends on what went wrong. This indicates a very unusual situation, and the user should contact technical support.

Version Compatibility

FSC performs a version compatibility check between the PRIMEFastSpellChecker macro and P6FAST.WLL. If they're not the same, you get this message

> **PRIME Fast Spell Check *n.nnn* found version *n.nnn* of P6FAST.WLL. Try reinstalling PRIME 6.**

If you get this message you may want to contact technical support.

File New

The PRIME FileNew utility replaces WinWord's default FileNew with a dialog box containing option buttons labeled with user-defined descriptions for standard templates. This eliminates scrolling through WinWord's template list looking for cryptic eight-character template names.

You can have fourteen custom option buttons in addition to some other built-in functions for creating templates or accessing WinWord's default FileNew command.

You can configure the PRIME FileNew dialog box to display a preview of the selected template.

Using PRIME FileNew is as simple as clicking on the option button that corresponds to the type of document you want to create, then clicking OK. A new document is created based on the template associated with the option button you selected.

The **Select a format for your new document** group box displays option buttons labeled with text describing each template. To create a new document using the template attached to a particular button, click on the button describing the template and then click on OK.

The **WinWord's default FileNew** push button invokes the default WinWord FileNew command. Use this for those rare occasions when you want to remember how awkward WinWord's default FileNew really is.

The **Create a new template** push button creates a new template based on your NORMAL.DOT template.

The **Open an existing template** push button displays a list of all files found in the two user-definable directories used to store templates and wizards.

All files from the "User Templates" and "Workgroup Templates" directories are listed. Normally, only template (.DOT) and wizard (.WIZ) files are found in these directories but the utility lists *all* files found. From this list select the file you want to open and click the **Open** button. If you have defined only one of the two possible directories (Tools Options File Locations), only the defined directory is listed.

Choosing the push button **Modify options on this menu** prompts you for the option button you want to change on the primary **PRIME FileNewUpdate** dialog box.

The PRIME FileNew utility is designed to have each option button associated with a different template. If you want to use the same template for more than one button you'll need to make a copy of that template and give it a different name.

You can change the description and the associated template for each of the option buttons without editing the macro code. To do so, use the **Modify options on this menu** command button.

Related Topics:

Modify options on this menu
FileNew Warnings
Network Considerations
MMASTER.INI -- PRIME's Private Settings File

Modify options on this menu

This dialog box is used to select an option button so that its settings can be changed.

After you select the option button you want to change, clicking on **Change** takes you to the update screen for that button.

When you have made changes for all the option buttons you want to modify click on the **Done** button.

This dialog box is also used to enable or disable the template preview feature of the FileNew dialog box. Check or clear the **Check here to [ ] enable template preview** check box as appropriate and click **Done**.

Related Topics:

Setting up an option button
Current description for this option button
Current template attached to this option button
Current save path attached to this template
Immediate save on file creation
Current filename mask attached to this template

Setting up an option button

When you choose an option button and click **Change** you see the update screen that displays the current settings for that button.

In the update screen click on the **Change** button for the setting you want to define or change. Each of the Change buttons take you to a screen for that setting except for the **Immediate save on file creation** change button. This setting is a simple toggle – clicking on its **Change** button sets it to Yes and clicking again sets it to No.

Once all you have made all your changes, click on the **Complete** button to affect the changes and return to the primary PRIME FileNewUpdate dialog box, where you can select another option button for modification if desired.

To delete all the settings for the current option button, click on the **Delete all** button. The description for the option button is automatically reset to "Reserved."

Current description for this option button

This setting is "Reserved" by default indicating that the option button has not been configured.

When you click the **Change** button for the "Current description for this option button" a panel appears showing you the current description for the option button you are changing (Old Option Button Description) and a text box where you can type in a new description for this option button. You can type up to 255 characters in the text box but only 20 or 40 characters display properly in the FileNew dialog box. This is because Word uses proportional fonts in its dialog boxes and you may see more or fewer characters depending on the description you type.

Once your changes are complete, click on the **Changes complete** button to affect the changes and return to the dialog box showing you all the settings for the current option button. If you want to return to this dialog box without keeping any changes, click on the **<Back** button.

Current template attached to this option button

This setting is required. It defines the template used to create a new document for this option button.

When you click the **Change** button for the "Current template attached to this option button" a panel appears in which you see the template currently attached to the option button (Current Template) and a combo box for selecting a template or wizard file. The list of files displayed comprise *all* files from the "User Templates" and "Workgroup Templates" directories as defined in Tools Options File Locations, not just templates and wizards. Only template (.DOT) or wizard (.WIZ) files should be attached to an option button.

You are not limited to using template and wizard files that reside in these two directories. One of PRIME FileNew's major benefits is that you can locate template and wizard files in any directory and attach them to an option button. However, if the template or wizard is not displayed in the list, *you must manually type in a fully qualified path and filename in the text box*. If the file name you type in does not exist, a message box prompts you to enter a valid path and filename.

Once you have picked a valid template or wizard click on the **Changes complete** button to affect the changes and return to the dialog box showing you all the settings for the current option button. If you want to return to this dialog box without keeping any changes, click on the **<Back** button.

Current save path attached to this template

This setting is not required.

Each template associated with an option button can be assigned a default save path. When a document (based on a template with a save path) is saved for the first time, the default save path is logged onto and displayed in the WinWord SaveAs dialog box. *You are not required to save the document to this path*. This default save path is simply provided as a convenience that you can override for any individual new file. To override the default save path, change the directory in the SaveAs dialog box.

When you click the **Change** button for the "Current save path attached to this template" a panel appears wherein you define the default save path for the template attached to this option button. You can (1) type in a fully qualified path in the text box or (2) use the Directories and Drives list boxes to "walk" your directory and drives to locate the desired directory.

You can have only one save path associated with any one template. If you want to use the same template for more than one FileNew option button you'll need to make a copy of that template and give it a different name.

Once a valid save path has been defined, click on the **Changes complete** button to affect the changes and return to the dialog box showing you all the settings for the current option button. If you want to return to this dialog box without keeping any changes, click on the **<Back** button.

Immediate save on file creation

This setting is No by default.

This setting reminds you to save each document you create by popping up the WinWord SaveAs dialog box when a new document is created via this option button.

Clicking on the **Change** button toggles this setting between Yes and No.

Current filename mask attached to this template

The filename mask settings allow you to create a default filename and/or extension for documents based on a given template.

When you click the **Change** button for the "Current filename mask attached to this template" a panel appears wherein you can set or change the filename mask for this template.

A filename mask can consist of two parts, the filename and the extension. The filename can be further divided into a filename mask and an automatic increment. This allows you to let FileNew create complete unique filenames for your documents.

The **Current filename mask** is optional (unless you enter an automatic incrementing counter value which makes a filename mask a requirement).

For example, you may want to have your initials be part of every contract you create. You could enter in the **Current filename mask** text box '**TJL**' (without quotes). When you create a new document based on the template attached to this option button and save the file, the default filename in the SaveAs dialog box would be:

TJL.DOC

To remind yourself to make the filename unique you might want to use '**TJLXXXXX**' as the filename mask. You would then change the X's to make the name unique.

The **Automatic increment** counter value is optional.

To create an incrementing counter, enter '**000**' (without the quotes) in the **Automatic increment** text box (or whatever three-digit number you want to increment from). The counter can only be used in conjunction with a filename mask. If you use a counter you must limit the filename mask to five or fewer characters. To continue the previous example you could use '**TJL**' as the filename mask and '**000**' as the increment. When you save the first document based on the template attached to this option button you will see this as the default filename in the SaveAs dialog box:

TJL001.DOC

Each counter increments by one for each subsequent document based on this template. The maximum counter value is 999. When this value is reached you must reset the counter for that template.

The **Custom extension** is optional.

If you get tired of every document's extension being '**.DOC**' by default you can enter a custom extension to be used for each document based on the template attached to this option button. For example, if you want all letters to have '**.LTR**' as a default extension, enter '**LTR**' (without quotes) in the **Custom extension** text box. When you initially save the document you will get a default filename in the SaveAs dialog box of:

.LTR

The **Custom extension** setting can be used by itself, or in conjunction with the filename mask or the filename mask/automatic increment settings.

Once your changes are complete, click on the **Changes complete** button to affect the changes and return to the dialog box showing you all the settings for the current option button. If you want to return to this dialog box without keeping any changes, click on the **<Back** button.

FileNew Warnings

Each time you choose FileNew from WinWord's menu bar the PRIME FileNew command reads the **MMASTER.INI** file to get the button descriptions and the associated template file names. If you delete the **MMASTER.INI** file, the command recreates it and your option buttons all revert back to "Reserved." If you delete a template that is associated with a PRIME FileNew button and then select that button, you will get a message box telling you that the template cannot be found and that you should reassign the option button to an existing template. If you select a reserved button you will get the error message "no TEMPLATE associated with this selection."

If you close a newly created document from the application control menu (**ALT+SPACEBAR+C** or **ALT+F4**), or by choosing Exit from the File menu you will bypass PRIME FileNew's ability to change the current directory to the default save path assigned to the template for that document. This does not affect your ability to save your documents, it just means the default save path (if one has been defined) and any filename mask information will not be found.

Network Considerations

If you are using PRIME on a network you should understand how its private INI file − called MMASTER.INI − is used. If MMASTER.INI does not exist, PRIME by default creates it in the directory that contains the WIN.COM command file (this is the file that starts Windows; on a stand-alone PC, WIN.COM is usually located in the C:\WINDOWS directory on the local hard disk). For information on other setting parameters for MMASTER.INI, see MMASTER.INI -- PRIME's Private Settings File.

If you are running Windows on a network, the WIN.COM program may be running from the network server's hard disk. If this is the case then there is probably only one MMASTER.INI file and more than one PRIME user may be saving information to it. Here's the solution to this dilemma.

1. Add the following lines to the end of each user's WIN.INI file:

   [PRIME]

   MMasterPath=c:\data\winword\docs\nancy\

2. This tells PRIME where to find a given user's MMASTER.INI file. The text following the equal sign should be the complete path where the MMASTER.INI file is stored *and must include the trailing backslash*. In the example shown, "c:\data\winword\docs\nancy" is where Nancy's MMASTER.INI will be stored. Use a path that's appropriate for your system.
3. The next time PRIME runs, it will detect this path setting and create a new MMASTER.INI in the specified directory.

We repeat − there must be one unique MMASTER.INI file stored in a separate directory for each network user.

MMASTER.INI -- PRIME's Private Settings File

This section describes the various MMASTER.INI settings.

[Templates]

This section is used exclusively by the File New command set.

EnablePreview=

This setting is either Yes or No and determines if the **PRIME FileNew** dialog box is displayed in template preview mode.

MenuName1=, MenuName2=, ... MenuName14=

These keys store the option button descriptions for the **PRIME FileNew** dialog box. An example is "A &Plain document".

Template1=, Template2=, ... Template14=

These are the full path and file names of each template defined for FileNew. An example is "Template1=c:\winapps\winword6\template\plain.dot". If no template is assigned then the value is blank.

ISTemplate1=, ISTemplate2=, ... ISTemplate14=

IS stands for "Immediate Save" and determines if a SaveAs is performed when a document is created. This setting is either Yes or No.

SPTemplate1=, SPTemplate2=, ... SPTemplate14=

SP stands for "Save Path" and controls the default save path for PRIME FileNew templates. For example, "SPTemplate1=c:\data\winword\plain". If there is no default save path then the value is blank.

FMTemplate1=, FMTemplate2=, ... FMTemplate14=

FM stands for "File Mask" and controls the default filename that appears in the SaveAs dialog box. It consists of three parts (separated by a semicolon and the pipe character) in this order:

filename **;** *extension* **|** *counter*

If a filename mask was created at one time for a given option button and is subsequently removed via FileNewUpdate this setting will appear like "FMTemplate1=;|". If there never was a filename mask setting then the value is blank.

Filename can consist of a number of alphanumeric characters (maximum of eight unless a counter is used in which case the maximum is five characters). *Extension* can consist of up to three alphanumeric characters. *Counter* (if used) must consist of three numeric digits between 000 and 999.

Macro Manager

Select for opening or printing one, several, or all macros from a list of the current template's macros.

Macro Manager lets you open or print any or all of the macros in the active template (the one that the File Templates dialog box lists next to the Attach button).

Note that Macro Manager does not list macros available to you from global add-in templates – only the ones in the active template.

To open one macro, simply select the macro from the list and click **Open a Macro**. You can also open a macro by double-clicking it. An open macro is surrounded by parentheses (like this) in the **Macros** list. You can open any number of macros this way. When you are done opening or printing macros, click **Cancel**.

To open all of the macros, click **Open All Macros**. Macro Manager opens all listed macros that are not already open. When it is done, the dialog box closes itself automatically.

To print one macro, simply select the macro from the list and click **Print a Macro**. You can print any number of macros this way. Macros are background printed: they print even while the dialog box remains displayed. When you are done printing macros, click **Cancel**.

To print all of the macros, click **Print All Macros**. Macro Manager prints all listed macros. When it is done, the dialog box closes itself automatically.

To access Word's Organizer dialog, click the **Organizer** button.

Related Topics:

Output Parameters

Output Parameters

Each macro is printed as a separate document with a heading that contains the template name, macro name, and printing date, and a footing that contains the page number.

Related Topics:

Page Breaks
Changing the Output Format

Page Breaks

You can insert a page break in a macro at any point with the following special "command":

```
'PAGE
```

The command will not appear in the output, nor will any blank lines that precede it.

Changing the Output Format

A printed macro's font, point size, line spacing, etc., are controlled by the **Macro Text** style in NORMAL.DOT. Its page layout is controlled by the page layout of NORMAL.DOT. NORMAL.DOT controls the appearance of a macro window as well, so your macro should look about the same when printed as it does when displayed on the screen. For information on a related bug in Word, see NORMAL.DOT Macro Text Style Misbehaves.

By default, a printed macro has a heading with the template and macro name on the left and the date of printing on the right, and a footer with the page number in the center. All are in the font used by the NORMAL style in the attached template (the template whose macros are being printed), with boldface.

To change the heading and footing, define AutoText entries named MacroHeading and MacroFooting. Define these AutoText entries in a template that contains macros to change the heading and footing used to print those macros. Define them in NORMAL.DOT to change the "default" heading and footing used to print macros from any template that does not have its own heading and footing definitions.

MacroHeading contains the text to appear in the heading on each page. MacroFooting contains the text to appear in the footing. Each may consist of a single paragraph of text, or several paragraphs.

You can use the following special codes in MacroHeading and MacroFooting to insert variable information in each printed heading or footing:

- **$t** to insert the name of the attached template (the template that contains the macro being printed).
- **$m** to insert the name of the macro being printed.
- **$d** to insert the description of the macro being printed, at it appears in the Tools Macro dialog box when you click the macro.
- **$$** to insert a single '$'. To insert '$t' in a macro, for example, put '$$t' in the AutoText entry.

You can also use fields to insert variable information such as printing date and page number.

Proof Controller

Can't find that elusive Tools Language "(no proofing)" dialog that forces the spell checker to skip a selection? Thus was born PRIME Proof Controller. Highlight a section of your document and toggle the language between "no proofing" (which excludes the section from spell checking) and the language of your choice.

Proof Controller displays the current selection's language. Below that, Proof Controller presents three options in the form of push buttons.

- Set selection to (no proofing)
- Set selection to default language
- Change a language for selection

The (no proofing) option prevents Word from hyphenating, spell checking, or grammar checking any such marked text.

ResetChar

Another handy tool for fixing what's broken in Word 6.0. Makes the **CTRL+Spacebar** combination reset the character format of the insertion point without resetting the format of the preceding word.

When *ResetChar* is present in any available template, Word runs it automatically when you enter **CTRL+Spacebar.** There ordinarily is no reason to run the macro from the **Tools Macro** command.

When you enter **CTRL+Spacebar** and Word runs *ResetChar,* you may notice that the part of the line after the insertion point "jogs" to the right, then back again. This is a harmless side effect of a trick that *ResetChar* must play to do its job.

If you don't like the effect of *ResetChar,* simply delete it from NORMAL.DOT after you install PRIME 6.

Related Topics:

Why Is ResetChar Useful?

Why Is *ResetChar* Useful?

When you enter **CTRL+Spacebar** with the insertion point in the middle of a word, Word 6 resets the character format of the entire word, not the insertion point alone.

Example: Suppose you're editing a document that contains this sentence:

> **This program is guaranteed to work under water.**

The program is named *Win*Glop, so you want to change the sentence to read:

> ***Win*Glop is guaranteed to work under water.**

To do this, you:

1. Move the insertion point to the start of the sentence.
2. Press **CTRL+SHIFT+** → twice, selecting the text from 'This' through the space following 'program'.
3. Enter **CTRL+I,** selecting italic formatting.
4. Type '*Win*'.
5. Press **CTRL+Spacebar** to reset character formatting.
6. Type 'Glop' and a space.

Whoops! At step 5, Word reset the character formatting of '*Win*' as well as that of the insertion point itself! And avoiding this "feature" turns out to be devilishly hard.

But *ResetChar* fixes that.

Toolbar Lister

Toolbar Lister provides on screen a complete list of all active toolbars by name, along with their slot numbers and assignments. You can catalog this list into a table.

The **Toolbar Names** drop-down list box shows active toolbar names listed in the order in which they appear in the native Toolbars dialog (not alphabetical). Selecting a toolbar other than Standard updates the **Button Contents** list box, which contains a list of each button slot and its current assignment (a blank represents a space or gap on the toolbar, which *does* occupy a slot). Button separator slots (slots occupied by a space instead of a button image) are represented as "<space>" in the utility's dialog box and the catalog.

The **Log All** button generates a three-column table in a scratch document (based on the current template) that you can then print or save.

Related Topics:

Table Column Headings

Table Column Headings

Toolbar Lister's log table column headings follow.

- Counter
- Toolbar Name
- Button Information

Window Manager

Select for closing one, several, or all open Word windows; and create a new window, split, and arrange windows.

To close one window, simply select the window from the list and click **Close a Window**. You can also close a window by double-clicking it. You can close any number of windows this way. When you are done closing windows, click **Cancel**.

To close all windows, click **Close All Windows**. Window Manager closes all listed windows, then ends automatically. This leaves Word in null menu mode, with no documents open and only File and Help in the menu bar.

New Window opens a new window with the same contents as the current window in the **Windows** list. This is equivalent to selecting Window then choosing New Window in Word's native menu.

Split a Window splits the current window in the Windows list into two equally sized horizontal panes. This is equivalent to selecting Window then choosing Split in Word's native menu. Clicking on **Split a Window** a second time has no effect.

Arrange Windows arranges all non-minimized windows. This is equivalent to selecting Window then choosing Arrange All in Word's native menu.

Zoomer

PRIME Zoomer makes it easy to control your document's zoom setting. You can control the zoom setting with a slider (like the controls on a stereo) or select it from customized lists of recent and fixed settings.

There are three ways to use the slider:

- Drag the thumb bar to the setting you want. Watch the **Zoom** control to see how the zoom will be set. When the zoom value is right, click OK.
- Click the slider to the left or right of the thumb bar to move the zoom setting down or up 1% at a time. Click OK.
- Double-click at any point on the slider to move the thumb bar to that point. Click OK.

There are other ways to change the zoom setting:

- Type a new value into the **Zoom** control. Click OK.
- Click an entry in the list of **Previous** settings or **Fixed** settings. Click OK.
- Double-click a **Previous** or **Fixed** entry. (You need not click OK.)
- Click **Page Width**, **Whole Page**, or **Two Pages**. (You need not click OK.)

Helpful hints:

- The **Customize** button opens the **Zoomer Customize** dialog box. This box controls aspects of PRIME Zoomer's operation such as the number of previous zoom settings in the **Previous** list and the contents of the **Fixed** list.
- The lists of **Previous** and **Fixed** settings are enabled only when there actually are previous settings and fixed settings to select.
- The slider scale shows the range of settings that all of the PRIME Zoomer controls accept. This may be more limited than the View Zoom command's range of zoom settings (10% to 200%). Limiting the range of the slider scale makes the slider more precise. Click the **Customize** button to change the slider's scale.

The **Previous** list box contains up to five of the most recent zoom values you have set with PRIME Zoomer. To change the number of values in the **Previous** list box, click the **Customize** button.

The **Fixed** list box contains up to five fixed zoom values that you use frequently. To add, delete, or change the values in the **Fixed** list box, click the **Customize** button.

The **Page Width** button sets the zoom to the largest value that allows the width of a page to appear in the active window.

The **Whole Page** button sets the zoom to the largest value that lets a whole page appear in the active window.

The **Two Pages** button sets the zoom to the largest value that lets two pages appear in the active window.

Related Topics:

Zoomer Customization
Zoomer Customization Error Messages
Calling Zoomer from Your Own Macro

Zoomer Customization

The **PRIME Zoomer Customize** dialog box lets you customize PRIME Zoomer's behavior.

Slider Range Minimum and **Maximum** set the range of the slider. These values also set limits on the values that may appear in the **Previous** and **Fixed Settings** lists, and the values you may enter in the **Zoom** control.

Previous Settings controls the number of previous settings that the **Previous** list will retain. It may be set to any value from 0 (an empty list) to 5.

Fixed Settings lets you specify up to five values to appear in the **Fixed** list. To make less than five values appear in the list, set one or more to 0%. The 0% values need not be the last ones. Non-zero values will appear in the **Fixed** list in the same order as in this dialog box.

If you enter an invalid value in any field, PRIME Zoomer will beep and display a message on the status line when you try to leave the field. For more detailed explanations of these errors, see Zoomer Customization Error Messages.

Zoomer Customization Error Messages

Minimum must be at least 10% & no greater than Maximum.

The value of **Slider Range Minimum** is invalid. It must be a number no smaller than 10 (10%) and no greater than the value of **Slider Range Maximum**.

Maximum must be no less than Minimum & no greater than 200%.

The value of **Slider Range Maximum** is invalid. It must be a number no smaller than the value of **Slider Range Minimum** and no greater than 200 (200%).

The number of previous settings must be 0 to 5.

The value of **Previous Settings** is invalid. It must be a number in the range 0 to 5. A value of 0 makes PRIME Zoomer's **Previous** list empty; you will not be able to select it.

A Fixed Setting must be at least as great as Minimum and no greater than Maximum, or zero.

One of the **Fixed Settings** is invalid. Each **Fixed Setting** must be a number no smaller than the value of **Slider Range Minimum** and no greater than the value of **Slider Range Maximum**.

The **Fixed Settings** may also be zero. A zero setting is inactive; it will not appear in PRIME Zoomer's **Fixed** list.

Calling Zoomer from Your Own Macro

To call Zoomer from your own macro, simply execute the command

```
PRIMEZoomerWLL
```

PRIMEZoomerWLL is implemented by a file named P6ZOOM.WLL. The PRIME 6 installation procedure installs this file in your Word for Windows startup directory, making a declaration unnecessary. If the file is not in the startup directory for any reason, you must declare it at the beginning of the macro that uses it. (In the example below, P6ZOOM.WLL is located in C:\WINDOWS.)

```
Declare Sub PRIMEZoomerWLL Lib "C:\WINDOWS\P6ZOOM.WLL"

Sub MAIN
.
.
.
End Sub
```

You may encounter the following WordBasic errors if you set up a call to PRIMEZoomerWLL incorrectly.

Error 116, Argument-count mismatch

You tried to run PRIMEZoomerWLL with one or more arguments, for example:

```
PRIMEZoomerWLL(1)          ' this is wrong

PRIMEZoomerWLL "50%"              ' and this is wrong
```

You must run PRIMEZoomerWLL with no arguments:

```
PRIMEZoomerWLL             ' this is right
```

Error 124, Unknown Command, Subroutine, or Function

1. P6ZOOM.WLL is not in the Word for Windows startup directory, and is not declared to be elsewhere. Therefore WordBasic does not recognize PRIMEZoomerWLL as a valid WordBasic command.
2. You misspelled the PRIMEZoomerWLL command.

Error 543, Unable to open specified library

1. P6ZOOM.WLL is not in the directory where the DECLARE statement says it should be.
2. One or both of two DLLs used by P6ZOOM.WLL cannot be found. The DLLs are CTL3D.DLL and CSLIDER.DLL. The PRIME 6 installation procedure installs both of these files in your Windows system directory (customarily C:\WINDOWS\SYSTEM). For a complete PRIME 6 file inventory see File Inventory.

You Can Call PRIME 6 Custom Functions and Subroutines

This section contains Application Programming Interface (API) specifications for the subroutines and functions stored in PRIME 6's global macro library PRIMELib.

Related Topics:

PRIMELib.fActiveTemplate$(pathname)
PRIMELib.fBinSearch(List$(), Count, val$)
PRIMELib.fBinSearchInit(List$(), Count, val$)
PRIMELib.fBool(n)
PRIMELib.fDeleteDocContents
PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)
PRIMELib.fDocDisplayState
PRIMELib.fGetBkmkAlphaNumber(TheBookmarkName$)
PRIMELib.fGetBkmkLocNumber(TheBookmarkName$)
PRIMELib.fGetDocProtectionState
PRIMELib.fIsNormalOpen
PRIMELib.fIsTableInSelection(SelMode, ScreenUpdatingMode)
PRIMELib.fIsValidToolbarName(Toolbar$, Context)
PRIMELib.fMacroPaneTest(Title$)
PRIMELib.fMax(x, y)
PRIMELib.fMin(x, y)
PRIMELib.fNullMenuState(Title$)
PRIMELib.fOpenForAppendAs1(f$, t$)
PRIMELib.fOpenForInputAs1(f$, t$)
PRIMELib.fOpenForOutputAs1(f$, t$)
PRIMELib.fPathFromParts$(Path$, File$)
PRIMELib.fUBoundNum(Array())
PRIMELib.fUBoundStr(Array$())
PRIMELib.fWindowType
PRIMELib.sCopyN(t(), s(), n)
PRIMELib.sCopyNR(t(), s(), target, first, last)
PRIMELib.sCopyS(t$(), s$(), n)
PRIMELib.sCopySR(t$(), s$(), target, first, last)
PRIMELib.sGetToolbarButtonInfo(Toolbar$, ButtonInfo$(), Context)
PRIMELib.sGetToolbarButtonInfoX(Toolbar$, ButtonInfo$(), Context)
PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)
PRIMELib.sMsgNumArray(s(), n, name$)
PRIMELib.sMsgStrArray(s$(), n, name$)
PRIMELib.sRedefineBookmark(DestinationBkmk$, NewText$)
PRIMELib.sRedimN(array(), old, new)
PRIMELib.sRedimS(array$(), old, new)

PRIMELib.fActiveTemplate$(*pathname*)

This function returns the name of the active window's template. If *pathname* is 0 it returns just the 8.3 filename; if *pathname* <> 0 it returns the fully qualified path and filename.

If the active window is an unnamed template (like Template1), if *pathname* is 0 the function returns the unnamed template's name; if *pathname* <> 0 it returns the current directory plus a trailing backslash but no 8.3 filename. If the active window is a macro inside an unsaved template (like Template1), the function returns values as described for the case of an unnamed template.

For additional information about what the current directory is, see the WordBasic help file or the *Word Developer's Kit* regarding **DefaultDir$()** and a *Type* value of 14.

See also

PRIMELib.fWindowType

PRIMELib.fBinSearch(*List$(), Count, val$*)

This function does a binary search and returns the 0-based index of the matching *List$()* element if a match is found (the search is case-sensitive), and returns a negative number if no match is found. In this case, to find the index where *val$* should be inserted, subtract the return value from -1.

List$() is an array containing a string list; the function assumes the list is sorted alphabetically in ascending order.

Count is the number of entries in the list (not necessarily the number of elements in the array).

val$ is the string to search for.

See also

PRIMELib.fBinSearchInit(List$(), Count, val$)

PRIMELib.fBinSearchInit(*List$(), Count, val$*)

Same as fBinSearch(), but searches for an initial match, e.g., a search for "xyz:" finds a match on "xyz:abc".

See also

PRIMELib.fBinSearch(List$(), Count, val$)

PRIMELib.fBool(*n*)

This function returns BASIC Boolean values of either 0 or 1. If *n* is 0 the function returns 0, if *n* <> 0 it returns 1. Even though a "true Basic TRUE" would be -1, 1 is more useful in many situations, e.g., DlgEnable.

PRIMELib.fDeleteDocContents

This Boolean function returns -1 if it successfully selected all of and deleted the current document's contents; otherwise it returns zero. This function assumes the caller has already checked the current document's protection status.

See also

PRIMELib.fGetDocProtectionState

PRIMELib.fNullMenuState(Title$)

PRIMELib.fDeleteFromArray(*Array$(), Size, DeletePt*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This function deletes an entry from a string array. *Array$()* is the target array, *Size* is the number of elements in the array (one greater than the upper boundary dimension), *DeletePt* is the 0-based index of the element to be deleted. The function returns *DeletePt* - 1 if the highest element was deleted, else it returns *DeletePt*; however, if the only remaining element was deleted, the function makes Array$(0) null.

See also

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopyS(t$(), s$(), n)

PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimN(array(), old, new)

PRIMELib.sRedimS(array$(), old, new)

PRIMELib.fDocDisplayState

This function returns 0 if Word is in null menu mode (meaning, no documents of any kind are open), 1 if the current child window is minimized, 2 for restored, and 3 for maximized.

See also

PRIMELib.fNullMenuState(Title$)

PRIMELib.fGetBkmkAlphaNumber(*TheBookmarkName$*)

This function returns 0 if there is any error condition, for example, the current window is a macro pane, the current document is protected, the current document contains no bookmarks, or the bookmark name itself is invalid or doesn't exist.

Otherwise, this function returns a positive integer that represents a one-based index of the bookmark's position in the alphabetic (name) order of all bookmarks in the current document.

See also

PRIMELib.fGetBkmkLocNumber(TheBookmarkName$)

PRIMELib.sRedefineBookmark(DestinationBkmk$, NewText$)

PRIMELib.fGetBkmkLocNumber(*TheBookmarkName$*)

This function returns 0 if there is any error condition, for example, the current window is a macro pane, the current document is protected, the current document contains no bookmarks, or the bookmark name itself is invalid or doesn't exist.

Otherwise, this function returns a positive integer that represents a one-based index of the bookmark's position in the locational order of all bookmarks in the current document.

This function overcomes a Word bookmark bug that's documented in Form Field Bookmarks Don't Exist According to SelInfo(29).

See also

PRIMELib.fGetBkmkAlphaNumber(TheBookmarkName$)

PRIMELib.sRedefineBookmark(DestinationBkmk$, NewText$)

PRIMELib.fGetDocProtectionState

This function returns 0 for an unprotected document, 1 for a protected document (protected for Revisions, Annotations, or Forms), and 9 for a macro pane. Note that return values 2-8 are reserved for future use. This function assumes the caller has already tested for null menu mode.

See also

PRIMELib.fNullMenuState(Title$)

PRIMELib.fIsNormalOpen

This function returns 0 if NORMAL.DOT is not in the window list, and returns its window number if it is open. (The help text for WordBasic's **Window()** function states, "Returns a number that corresponds to the position of the active window on the Window menu, where 1 corresponds to the first position, 2 to the second position, and so on.") This function fIsNormalOpen assumes the caller has already tested for null menu mode.

See also

PRIMELib.fNullMenuState(Title$)

PRIMELib.fIsTableInSelection(*SelMode, ScreenUpdatingMode*)

This function returns -1 if there's a table in the selection, 0 if there is no table in the selection, or 9 if there's an error (see below). If *SelMode* is 1, the function scans the current selection only; if *SelMode* is 2, the function scans the entire current document; any other value of *SelMode* produces an error message box and the function returns 9. If *ScreenUpdatingMode* is <> 0 then at the end of the function it resets (activates) screen updating. If *ScreenUpdatingMode* is 0 then the function does not activate screen updating upon completion.

By design, this function may in some cases move the current document's insertion point and thereby change the selection. It's the caller's responsibility to capture the selection location before calling **fIsTableInSelection()**, then immediately afterwards reset the selection to its original state. You could use a throw-away bookmark to do this. You could also use the built-in functions **GetSelStartPos()** and **GetSelEndPos()** along with the command **SetSelRange** to do this.

PRIMELib.fIsValidToolbarName(*Toolbar$, Context*)

This function returns -1 if *Toolbar$* is a valid toolbar name for *Context*, 0 if it isn't a valid name. *Context* of 0, according to WordBasic's help file, represents "the toolbars that are available when a document based on the Normal template is active." *Context* of 1 represents "the toolbars that are currently available."

See also

PRIMELib.sGetToolbarButtonInfo(Toolbar$, ButtonInfo$(), Context)

PRIMELib.sGetToolbarButtonInfoX(Toolbar$, ButtonInfo$(), Context)

PRIMELib.fMacroPaneTest(*Title$*)

This function returns -1 if the active window is a macro pane and uses *Title$* for the resulting message box's title. The function returns 0 (no message) if the active window is not a macro pane. The message reads "Sorry, this macro cannot be run from inside the macro pane. Please switch to a document or template and try again." along with an OK button only and an attention icon.

PRIMELib.fMax(*x, y*)

This function returns the maximum of *x* and *y*.

See also

PRIMELib.fMin(x, y)

PRIMELib.fMin(*x, y*)

This function returns the minimum of *x* and *y*.

See also

PRIMELib.fMax(x, y)

PRIMELib.fNullMenuState(*Title$*)

This Boolean function returns -1 if Word is in the null menu state, i.e., no documents of any kind are open, and uses *Title$* for the resulting message box's title. The function returns 0 (no message) if Word is not in the null menu state. The message reads "Sorry, this macro cannot be run when no documents are open. Please open at least one document and try again." along with an OK button only and an attention icon.

See also

PRIMELib.fDocDisplayState

PRIMELib.fOpenForAppendAs1(*f$, t$*)

This Boolean function returns -1 if it successfully opened the file *f$* on channel 1 for updating. Otherwise it returns 0 and uses *t$* for the resulting message box's title. The message reads "The file ... cannot be opened for updating. The disk may be full, or the file may be read-only or locked by another application." along with an OK button only and an attention icon.

See also

PRIMELib.fOpenForInputAs1(f$, t$)

PRIMELib.fOpenForOutputAs1(f$, t$)

PRIMELib.fOpenForInputAs1(*f$, t$*)

This Boolean function returns -1 if it successfully opened the file *f$* on channel 1 for input. Otherwise it returns 0 and uses *t$* for the resulting message box's title. The message reads "The file ... cannot be opened for input. It may have been deleted or moved, or may be locked by another application." along with an OK button only and an attention icon.

See also

PRIMELib.fOpenForAppendAs1(f$, t$)

PRIMELib.fOpenForOutputAs1(f$, t$)

PRIMELib.fOpenForOutputAs1(*f$, t$*)

This Boolean function returns -1 if it successfully opened the file *f$* on channel 1 for output. Otherwise it returns 0 and uses *t$* for the resulting message box's title. The message reads "The file ... cannot be opened for output. The disk may be full, or the file may be read-only or locked by another application." along with an OK button only and an attention icon.

See also

PRIMELib.fOpenForAppendAs1(f$, t$)

PRIMELib.fOpenForInputAs1(f$, t$)

PRIMELib.fPathFromParts$(*Path$, File$*)

This function assembles a complete path name from its parts, consisting of the path, a backslash if necessary, and the file. *Path$* is the pathname, which may be null. *File$* is the 8.3 filename. This function does not validate *Path$* or *File$* in any way. That is the caller's responsibility.

PRIMELib.fUBoundNum(*Array()*)

This function returns the 0-based upper bound of the incoming numeric array, and returns -1 if there is any error.

See also

PRIMELib.fUBoundStr(Array$())

PRIMELib.fUBoundStr(*Array$()*)

This function is the same as fUBoundNum() except it's for string arrays.

See also

PRIMELib.fUBoundNum(Array())

PRIMELib.fWindowType

This function returns a value representing the type of entity occupying the current child window: 0 for a document, 1 for a template, and 2 for a macro.

See also

PRIMELib.fWindowType

PRIMELib.sCopyN(*t(), s(), n*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine is the same as sCopyS() except it's for numeric arrays.

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopyS(t$(), s$(), n)

PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimN(array(), old, new)

PRIMELib.sRedimS(array$(), old, new)

PRIMELib.sCopyNR(*t(), s(), target, first, last*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine is the same as sCopySR() except it's for numeric arrays.

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyS(t$(), s$(), n)

PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimN(array(), old, new)

PRIMELib.sRedimS(array$(), old, new)

PRIMELib.sCopyS(*t$(), s$(), n*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine copies an entire string array *s$()* − or only the initial *n* members of the array − to a separate target array *t$()*. The subroutine assumes *t$()* has already been properly dimensioned.

A simple example of a call to sCopyS() would look like this.

```
Sub MAIN
Count = 10                ' absolute count (ubound = Count - 1)
Dim Source$(Count - 1)
' Fill Source$() however you choose
Dim Target$(Count - 1)
PRIMELib.sCopyS Target$(), Source$(), Count
End Sub
```

An example using our fUBoundStr() shows how to handle a situation where you don't necessarily know Source$()'s upper bound at a given point in your code.

```
Sub MAIN
Count = 10                ' absolute count (ubound = Count - 1)
Dim Source$(Count - 1)
' Fill Source$() however you choose
' Lots of intervening stuff goes here ...
'    and now you need to know Source$()'s upper bound
n = PRIMELib.fUBoundStr(Source$())
Dim Target$(n)
PRIMELib.sCopyS Target$(), Source$(), n + 1
End Sub
```

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimN(array(), old, new)

PRIMELib.sRedimS(array$(), old, new)

PRIMELib.sCopySR(*t$(), s$(), target, first, last*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine copies a range of elements in a source string array *s$()* to a separate target array *t$()* without impacting elements in *t$()* above or below the range defined by *first..last*. The subroutine assumes *t$()* has already been properly dimensioned. The remaining arguments are described as follows.

- *target* – the 0-based index of the target element to receive the first copied element
- *first* – the 0-based index of the first *s$()* element to copy
- *last* – the 0-based index of the last *s$()* element to copy

A simple example of a call to sCopySR() would look like this, where the first and only the first element in the source array is being copied to the first element in the target array. All remaining elements in the target array are left intact.

```
Sub MAIN
Count = 10                  ' absolute count (ubound = Count - 1)
Dim Source$(Count - 1)
' Fill Source$() however you choose
Dim Target$(Count - 1)
' Fill Target$() with stuff, then use sCopySR to ...
'    poke in just one source element to element 0,
'    leaving all other Target$() elements as is
PRIMELib.sCopySR Target$(), Source$(), 0, 0, 0
End Sub
```

An example using our fUBoundStr() shows how to handle a situation where you don't necessarily know Source$()'s upper bound at a given point in your code.

```
Sub MAIN
Count = 10                  ' absolute count (ubound = Count - 1)
Dim Source$(Count - 1)
' Fill Source$() however you choose
' Lots of intervening stuff goes here ...
'    and now you need to know Source$()'s upper bound
n = PRIMELib.fUBoundStr(Source$())
Dim Target$(n)
PRIMELib.sCopyS Target$(), Source$(), n + 1
End Sub
```

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopyS(t$(), s$(), n)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimN(array(), old, new)

PRIMELib.sRedimS(array$(), old, new)

PRIMELib.sGetToolbarButtonInfo(*Toolbar$, ButtonInfo$(), Context*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine redimensions *ButtonInfo$()* to contain button information strings for each slot of the toolbar named by *Toolbar$*. Each string is comprised of three concatenated items: (1) button number (1-based), (2) " - " (space hyphen space), and (3) category item name. The function validates *Toolbar$* based on *Context* and redimensions *ButtonInfo$()* to 0 if *Toolbar$* is bogus. If *Toolbar$* is valid but contains no buttons, the function sets ButtonInfo$(0) to the string "The toolbar " + Toolbar$ + " has no buttons!" If a slot is a space, the subroutine uses the string "<space>" as the category item name.

See also

PRIMELib.fIsValidToolbarName(Toolbar$, Context)

PRIMELib.sGetToolbarButtonInfoX(Toolbar$, ButtonInfo$(), Context)

PRIMELib.sGetToolbarButtonInfoX(*Toolbar$, ButtonInfo$(), Context*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine updates *ButtonInfo$()* to contain button information strings for each slot of the toolbar named by *Toolbar$*. The subroutine assumes *ButtonInfo$()* has been properly dimensioned before the call. Each string is comprised of three concatenated items: (1) button number (1-based), (2) " - " (space hyphen space), and (3) category item name. The function validates *Toolbar$* based on *Context* and sets all elements of *ButtonInfo$()* to empty strings if *Toolbar$* is bogus. If *Toolbar$* is valid but contains no buttons, the function sets ButtonInfo$(0) to the string "The toolbar " + Toolbar$ + " has no buttons!" If a slot is a space, the subroutine uses the string "<space>" as the category item name.

See also

PRIMELib.fIsValidToolbarName(Toolbar$, Context)

PRIMELib.sGetToolbarButtonInfo(Toolbar$, ButtonInfo$(), Context)

PRIMELib.sInsertInArray(*Array$(), Size, InsertPt, Element$*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine inserts an entry in a string array. *Array$()* is the target array, *Size* is the number of elements in the array (one greater than the upper boundary dimension), *InsertPt* is the 0-based index of the element before which the entry is to be inserted, and *Element$* is the string to be inserted.

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopyS(t$(), s$(), n)

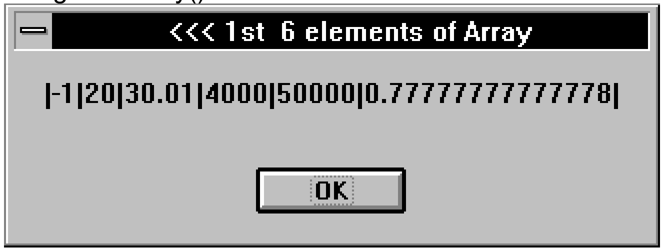
PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sRedimN(array(), old, new)

PRIMELib.sRedimS(array$(), old, new)

PRIMELib.sMsgNumArray(*s(), n, name$*)

This subroutine is intended for use as a debugging tool to display the first *n* members of a numeric array *s()* named *name$*. If the length of the resulting message box string exceeds the maximum allowable 255 characters, a WordBasic error 513 "String too long" occurs. A sample message box produced by sMsgNumArray() follows.



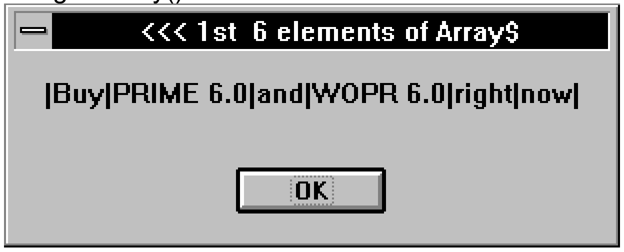
```
<<< 1st  6 elements of Array

|-1|20|30.01|4000|50000|0.77777777777778|
```

See also

PRIMELib.sMsgStrArray(s$(), n, name$)

PRIMELib.sMsgStrArray(*s$(), n, name$*)

This subroutine is intended for use as a debugging tool to display the first *n* members of a string array *s$ ()* named *name$*. If the length of the resulting message box string exceeds the maximum allowable 255 characters, a WordBasic error 513 "String too long" occurs. A sample message box produced by sMsgStrArray() follows.



```
|Buy|PRIME 6.0|and|WOPR 6.0|right|now|
```

See also

PRIMELib.sMsgNumArray(s(), n, name$)

PRIMELib.sRedefineBookmark(*DestinationBkmk$, NewText$*)

This subroutine redefines *DestinationBkmk$* to contain *NewText$*. It is well-behaved regardless of the current Typing Replaces Selection, Smart Cut and Paste, and Overtype settings (Tools Options Edit). In other words, it operates independently of these editing options. Furthermore, it leaves these settings as it found them before it was called. It is well-behaved for all combinations of *NewText$* and the contents of *DestinationBkmk$*:   null to null, null to not null, not null to null, and not null to not null. (Note that here "null" means "empty," not an ASCII 0.)

We wrote this subroutine because of the way Word responds to simple attempts to insert text into an existing bookmark under certain conditions. If you use the EditGoTo command followed by the Insert command, this actually destroys the destination bookmark *depending on the current Tools Options Edit settings*. In Word 6.0a this is not a bug, rather it is a byproduct of the Typing Replaces Selection setting which, when on, deletes any bookmark(s) in the selection when you replace that selection with new text.

There are 32 different cases that need to be tested when dealing with an allegedly edit option-independent bookmark redefinition procedure. First, there are eight combinations of Typing Replaces Selection (RS), Smart Cut and Paste (SCP), and Overtype (OT).

1. RS off, SCP off, OT off
2. RS on, SCP off, OT off
3. RS off, SCP on, OT off
4. RS on, SCP on, OT off
5. RS off, SCP off, OT on
6. RS on, SCP off, OT on
7. RS off, SCP on, OT on
8. RS on, SCP on, OT on

Then there are four possible source and destination bookmark conditions.

1. Empty source redefines an empty destination
2. Empty source redefines a non-empty destination
3. Non-empty source redefines an empty destination
4. Non-empty source redefines a non-empty destination

Eight times four is 32 total cases. Let's just look at what happens in each of the eight edit option combinations when a non-empty source redefines a non-empty destination. When you examine the following table's "Bkmk Contents After" column you'll see why you need a custom procedure like sRedefineBookmark() that preserves the original bookmark while redefining its contents. Assume the destination bookmark called Bookmark1 includes the text "Hello world." and that the text to be inserted is "NEW TEXT" using the following two-line macro.

Note that Word destroys the bookmark regardless of the bookmark's position in the document – (1) bookmark's head is at the start of the document, (2) bookmark's tail is at the end of the document, and (3) neither case is true and the bookmark is "mid-stream" in the document.

```
Sub MAIN

EditGoTo "Bookmark1"

Insert "NEW TEXT"

End Sub
```

	Bkmk Contents	Bkmk Contents

Edit Options	Before	After
RS off, SCP off, OT off	Hello world.	NEW TEXTHello world.
RS on, SCP off, OT off	Hello world.	<Destroyed>
RS off, SCP on, OT off	Hello world.	NEW TEXTHello world.
RS on, SCP on, OT off	Hello world.	<Destroyed>
RS off, SCP off, OT on	Hello world.	NEW TEXTHello world.
RS on, SCP off, OT on	Hello world.	<Destroyed>
RS off, SCP on, OT on	Hello world.	NEW TEXTHello world.
RS on, SCP on, OT on	Hello world.	<Destroyed>

If you perform this experiment again with the macro shown below that uses our custom sRedefineBookmark(), the bookmark is preserved in all eight cases shown in the table.

```
Sub MAIN

PRIMELib.sRedefineBookmark("Bookmark1", "NEW TEXT")

End Sub
```

See also

PRIMELib.fGetBkmkAlphaNumber(TheBookmarkName$)

PRIMELib.fGetBkmkLocNumber(TheBookmarkName$)

PRIMELib.sRedimN(*array(), old, new*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine redimensions a numeric array non-destructively, where *old* is the old upper boundary dimension (1 less than the number of elements) and *new* is the new upper boundary dimension.

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopyS(t$(), s$(), n)

PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimS(array$(), old, new)

PRIMELib.sRedimS(*array$(), old, new*)

Word bug alert ... see ReDim'ing a Passed Array Fails Intermittently.

This subroutine is the same as sRedimN() except it's for string arrays.

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopyS(t$(), s$(), n)

PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimN(array(), old, new)

Word and WordBasic Bugs and Workarounds

The following sections contain information about the bugs we discovered while developing PRIME 6.

Related Topics:

AddInState() Better Documentation

This section doesn't a cover a bug per se. But the *Word Developer's Kit* and the help file could have done a better job laying out in a sensible fashion the meanings of each of the eight return values for AddInState(). See the table below.

- The "(1) Loaded" column indicates whether or not the file is currently loaded.
- The "(2) WLL" column indicates whether or not the file is a WLL. If it is not a WLL, then it must be a template (even though the extension may be something other than "DOT").
- The "(3) Auto-loads" column indicates whether or not the file is in the \Startup directory as specified in Tools Options File Locations. Any DOT or WLL in \Startup is loaded automatically when Word starts. Interestingly, if you have a legitimate template in \Startup but it doesn't have a DOT extension, you'll have to manually Add it via File Templates. Once you've done this one time, every time Word starts subsequently this template-without-DOT-extension does appear in the Global Templates and Add-ins list, *but it will never be checked automatically so you'll have to do it yourself (load it), every time*. Moral of the story: stick to DOT extensions for global templates (this may seem obvious but you won't find a discussion of this in the Microsoft documentation).

Return Code	**(1) Loaded**	**(2) WLL**	**(4) Auto-loads**	**Description**
0	No	No	No	Not loaded now, not a WLL, not in \Startup.
1	Yes	No	No	Loaded now, not a WLL, not in \Startup.
2	No	Yes	No	Not loaded now, is a WLL, not in \Startup.
3	Yes	Yes	No	Loaded now, is a WLL, not in \Startup.
4	No	No	Yes	Not loaded now, not a WLL, is in \Startup.
5	Yes	No	Yes	Loaded now, not a WLL, is in \Startup.
6	No	Yes	Yes	Not loaded now, is a WLL, is in \Startup.
7	Yes	Yes	Yes	Loaded now, is a WLL, is in \Startup.

Bookmarks (Sort of) in Header/Footer Pane

Thanks to Steve Gerber for pointing out that Word 6 partially supports bookmarks in a header/footer pane.

Word 6 indeed allows you to think you've defined a valid bookmark inside a header/footer pane. If you've got Tools View Bookmark checked, you'll see it inside the pane when the header/footer pane is open. But beyond that, the rules appear inconsistent. The rules as we see them are ...

- When the header/footer pane is open, the native Edit Go To dialog [Go to What / Bookmark] drop-down list does *not* list the header/footer species of bookmark.
- When the header/footer pane is open, the native Edit Bookmark dialog *does* see the bookmark.

- Regardless of the pane display state of the current document, CountBookmarks() does not recognize any header/footer bookmarks, i.e., it does not include them in its count so if you've got no bookmarks in the document area and one bookmark in the header/footer pane, CountBookmarks() returns 0, that's right, *zero*.
- Regardless of the pane display state of the current document, a statement like EditGoTo .Destination = Bookmark$ produces error 1582 "Word cannot find the requested bookmark."

Call Statement Documentation Is Wrong

Microsoft's documentation on the Call statement says, "You cannot use the Call keyword to call a subroutine and pass arguments by value." *This is wrong*.

Run their own example from page 85 (example at the bottom of the page) in the *Word Developer's Kit* using Call and you'll see that Call doesn't disrupt the pass by value in any way. The message box dutifully, and correctly, reports "Hello" as the contents of **greeting$**. In other words, **greeting$** *does* get passed by value in the case when using the Call statement.

```
Sub MAIN
greeting$ = "Hello"
Call ChangeGreeting(greeting$)
MsgBox greeting$
End Sub
Sub ChangeGreeting(change$)
change$ = "What's up?"
End Sub
```

Document Variable Name Maximum Length Not Documented

The maximum length of a document variable name is not documented by Microsoft. The answer is – 255 characters. Try this.

```
Count = 255
VariableName$ = String$(Count, "*")
If Not SetDocumentVar(VariableName$, \
"vbl name's character count is" + \
Str$(Len(VariableName$))) Then
MsgBox "Could not add " + VariableName$
' Insert "Could not add " + VariableName$
End If
```

When **Count** is between 1 and 255, no problem. However, changing **Count** to 256 or greater causes an error 513 "String too long." The 513 error is not due to **VariableName$** exceeding a message box's 255 character maximum. You can substitute the statement

```
Insert "Could not add " + VariableName$
```

and Word still returns a 513 error (the interpreter points to **If Not...** as the offending statement).

EditAutoText Ignores AutoText Name's Maximum Length

AutoText names can be up to 32 characters but if you supply **EditAutoText** with a name greater than 32 characters it just ignores those beyond the 32nd with no error. For example,

```
Count = 255

EditAutoText .Name = String$(Count, "a"), .Context = 1, .Add
```

produces, without any error, an AutoText entry named "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa." However, changing **Count** to 256 or greater causes an error 513 "String too long."

Form Field Bookmarks Don't Exist According to SelInfo(29)

We have discovered that SelInfo(29) returns 0 for form field bookmarks, *when it should return a given bookmark's position in the document* (1 for the first bookmark, 2 for the second, and so on). Depending on how you interpret Microsoft's documentation on this function call, this 0 return value is either correct or a bug.

The tricky part is interpreting Microsoft's claim that the function call returns the "... number of the bookmark enclosing the start of the selection; 0 (zero) if none or invalid." In our opinion, a text form field's bookmark *does* enclose the start of the selection, but neither a check box form field's bookmark nor a drop-down form field's bookmark appear to. So why doesn't at least a text form field's bookmark come back as a non-zero value for SelInfo(29)? As soon as we get an answer from Microsoft, we'll let you know.

Moral of the story: use PRIME 6's custom function fGetBkmkLocNumber() to get back a bookmark's position, guaranteed. (See PRIMELib.fGetBkmkLocNumber(TheBookmarkName$).)

GetAutoText$() Always Returns Contents of First Duplicate-name Global AutoText

CountAutoTextEntries() and **AutoTextName$()** both include duplicate AutoText names that exist at the global context level. For example, if both NORMAL.DOT and a global add-in template AT.DOT contain an AutoText named Citibank, that name appears twice in the global name list.

But **GetAutoText$()** always returns the contents of the first duplicate-name global AutoText; it can't retrieve the contents of the second or subsequent duplicate-name global AutoTexts. For example, if NORMAL.DOT's Citibank contains "blue cheese" and AT.DOT's Citibank contains "blue moon" then **GetAutoText$("Citibank", 0)** returns "blue cheese" no matter what.

Workaround – Use PRIME 6 AutoText Lister to examine (and log, if desired) any duplicate-name global AutoText entries.

Incorrect Error Flag Value When Selection$() is Called from a WLL and Fails

When **Selection$()** is called from a WLL and fails, it returns a null string even though it neglects to set the "Too Much Data" flag.

Workaround – By testing the flag *and* the length of the return value, you can catch the failure.

Multi-Dimensional String Arrays Not Allowed in Custom Dialog Boxes

In Word 2 you could use a multi-dimensional string array in association with custom dialog box list controls, but not in Word 6. When the Word 6 interpreter encounters a list control definition statement that refers to a multi-dimensional string array, it produces a "Type mismatch" error 122.

Native Spell Checker Word Division Idiosyncrasies

Some punctuation characters are treated as word divisions whether they appear in an appropriate context or not. Others are apparently ignored.

Dog(mouse is accepted as two correctly spelled words, although *dogmouse* is recognized as misspelled. *Wish.bone* is treated as a misspelling, suggesting *wishbone;* and *outer.space* is treated as a misspelling suggesting nothing, even though *outer* and *space* are both valid words.

Dogfood(and *(dogcratic* and *.dogmouse* are all misspelled, but the parenthesis or period is not shown as part of the misspelling. *Dogwood(* and *(dogmatic* and *.doghouse* are all correctly spelled; the punctuation character is ignored.

NORMAL.DOT Macro Text Style Misbehaves

The *Word Developer's Kit* says that the format of a macro is determined by the definition of the Macro Text style in NORMAL.DOT. It doesn't say so, but the page width and left and right margins used to control line wrapping in a macro also appear to be determined by NORMAL.DOT.

However, when these parameters are changed in NORMAL.DOT, the effect on an existing macro is strange. When you open the macro there is no effect at all. If you print the macro, though, the change *does* affect the printed output. If you print the macro, then perform an editing operation (e.g., deleting the first line of the macro), then perform an Undo, the change affects the display as well.

NormalViewHeaderArea Command Broken for Footers in Page View

The command "NormalViewHeaderArea .Type=n" is supposed to place the insertion point in the header or footer, depending on the value of n and the settings of FilePageSetUp .FirstPage and .OddAndEvenPages. In normal view it should open a header/footer pane; in page view it should move the insertion point to the header or footer area of the page.

The command appears to work correctly in normal view, and for the header in page view. It works incorrectly for the footer (for n=1) in page view, placing the insertion point in the text area instead of the footer area.

ReDim'ing a Passed Array Fails Intermittently

WordBasic 6 took a big step forward by providing support for intra-macro and inter-macro passed arrays, something not possible in WordBasic 2. Unfortunately, the "redimension passed array" bug described in this section seriously undermines the effectiveness of this new capability.

This bug was first discovered and documented by the brilliant and indefatigable Scott Krueger. (Vince Chen, equally endowed, also played an important role in the bug's revelation and an elegant workaround. Jorge Vismara first cited and reported it up on PROGMSA.)

Scott's definitive public message on PROGMSA states, "If you ReDim in a function [or subroutine], they [the Word developers] are allocating a new array handle. This new handle is not getting used by the original calling function. 90% of the time, the new handle points to the same location in memory, so the function works. Intermittently, it points to a new location, and then the old handle is invalid."

As has been discussed on PROGMSA, one workaround is to use global arrays but this doesn't do you any good if the array needs to be passed outside the current macro. Vince adopted the approach of breaking procedures into two parts, "1) to return the size of the array, 2) to fill the array. Get the size first and Dim it, then pass the array to the function [or subroutine] to fill it."

Note that this bug does not prevent or restrict a called procedure from modifying one or more elements in a passed array. *The bug only manifests itself when the called procedure ReDim's the array*.

Before we had noticed this bug, we developed the procedures fDeleteFromArray(), sGetToolbarButtonInfo(), sInsertInArray(), sRedimN(), and sRedimS() that depend on stable behavior when ReDim'ing passed arrays. We've put together workarounds for all of them, discussed below, while leaving the original "way it should work and is documented to work by Microsoft" versions in the library, along with the (hopefully) temporary workarounds. Please go up on PROGMSA and cast your vote that this bug warrants immediate attention. Thanks. (A complete list of jump topics for all these procedures appears at the end of this section.)

- fDeleteFromArray() – Temporarily use sCopySR() to copy two disparate ranges up to and/or beyond, but not including, a specific element or elements.
- sGetToolbarButtonInfo() – Temporarily use sGetToolbarButtonInfoX().
- sInsertInArray() – Temporarily use sCopySR.
- sRedimN() – Temporarily use sCopyN and/or sCopyNR.
- sRedimS() – Temporarily use sCopyS and/or sCopySR.

See also

PRIMELib.fDeleteFromArray(Array$(), Size, DeletePt)

PRIMELib.sCopyN(t(), s(), n)

PRIMELib.sCopyNR(t(), s(), target, first, last)

PRIMELib.sCopyS(t$(), s$(), n)

PRIMELib.sCopySR(t$(), s$(), target, first, last)

PRIMELib.sGetToolbarButtonInfo(Toolbar$, ButtonInfo$(), Context)

PRIMELib.sGetToolbarButtonInfoX(Toolbar$, ButtonInfo$(), Context)

PRIMELib.sInsertInArray(Array$(), Size, InsertPt, Element$)

PRIMELib.sRedimN(array(), old, new)

PRIMELib.sRedimS(array$(), old, new)

Word Hangs After Creation of Huge Document Variable

In Word 6.0, if you set a document variable to a huge string (65,280 bytes) in an empty document, the document saves OK but subsequent activity (like tabbing from Word to File Manager) hangs Word – not a GPF but an application hang requiring an application reboot. In Word 6.0a, the same experiment produces a WINWORD.EXE GPF while the document is being saved.

Workaround – Don't write such a large string to a document variable, and test upper practical limits for document variable length for each individual system, since the error may be resource dependent.

Glossary of Terms

Active Template

The active template is the template to which the current document (or macro) belongs. Typically this is the template that originally spawned the document, but it is possible to attach a document to a different template using the File Templates command. Synonymous with the term "attached template."

Attached Template

The attached template is the template to which the current document (or macro) belongs. Typically this is the template that originally spawned the document, but it is possible to attach a document to a different template using the File Templates command. Synonymous with the term "active template."

AutoText

An AutoText entry (referred to in earlier versions of Word as glossary entry) is a chunk of boilerplate text, graphics, or combination thereof stored either in the current template or in NORMAL.DOT.

Document Variable

A document variable is a protected variable stored in the binary header of a document. "Protected" in the sense that you can only access document variables with WordBasic commands (or PRIMEDocVarManager, of course) as opposed to Word's user interface.

Global Macro Library

A macro stored in the global context layer (either in NORMAL.DOT or an add-in template) that contains a library of functions and subroutines callable by other macros.

Null Menu

Word is in null menu mode when there are no documents of any kind open, and the only commands displayed in the primary menu bar are File and Help.

WOPR

Woody's Office POWER Pack, a collection of Word add-ons from Pinecliffe International. To place an order for WOPR 6, call 800-659-4696 (or 314-965-5630 outside the US). For additional information call 314-965-5630.