

ewl shareres, T3EWCLASS, \$-!fJumpToHelpFile(hWndApp%c`wrdbasic.hlp%c0)\$dllres:wordres.dll:

HWBFY Programming with

Microsoft Word

Word Help Contents

WordBasic Statements and Functions

Conventions

Statements and Functions by Category

Statements and Functions Index

More WordBasic Information

What's New in WordBasic

Creating Dynamic Dialog Boxes

Key Examples in WordBasic Help

Operators and Predefined Bookmarks

Error Messages

Toolbar Button Images and Numbers

Converting Word Version 2.x Macros

The Microsoft Word Developer's Kit

Microsoft Solution Providers

Using special characters in a document search

In a FileFind instruction, you can use special characters to search for documents by using approximate criteria, instead of exact criteria. When you specify search criteria, you can use wildcard characters and search operators to control a search in specific ways.

Character	Meaning
? (question mark)	Match any single character. For example, specify "gr?y" to match both "gray" and "grey."
* (asterisk)	Match any number of characters. For example, specify "*.txt" to find all files that have the .TXT extension.
" " (quotation marks, Chr\$(34))	Matches all the characters, including spaces or punctuation, within the quotation marks. For example, specify Chr\$(34) + "modern dance" + Chr\$(34) to find the phrase modern dance.
\ (backslash)	Treats the following character (space, asterisk, question mark, comma, ampersand, or tilde) as a normal character. For example, specify "\?" to indicate a true question mark.
, (comma)	Logical OR. The information may match any or all items, but it must match at least one item. For example, specify "dance, modern" to find all documents that contain either "dance" or "modern."
& (ampersand) or (space)	Logical AND. The information must match all of the items in the list. For example, specify "dance&modern" or "dance modern" to find all documents that contain both words.
~ (tilde)	Logical NOT. The information must not match this item. For example, specify "~modern" to exclude files that contain the word "modern."

See also

[FileFind](#)

[Advanced search criteria](#)

Advanced search criteria

If you set `.PatternMatch` to 1 in a `FileFind`, `EditFind`, or `EditReplace` instruction, you can specify the following advanced search criteria.

To find	Operator	Examples
Any single character	<code>?</code>	"s?t" finds "sat," "set," and "sit."
Any string of characters	<code>*</code>	"s*d" finds "sad," "started," and "said."
One of the specified characters	<code>[]</code>	"w[io]n" finds "win" and "won."
Any single character in this range	<code>[-]</code>	"[r-t]ight" finds "right," "sight," and "tight." Ranges must be in ascending order.
Any single character except the characters inside the brackets	<code>[!]</code>	"m[!a]st" finds "mist," "most," and "must," but not "mast." "t[!ou]ck" finds "tack" and "tick," but not "tock" or "tuck."
Any single character except characters in the range inside the brackets	<code>[!x-z]</code>	"t[!a-m]ck" finds "tock" and "tuck," but not "tack" or "tick."
Exactly n occurrences of the previous character or expression	<code>{n}</code>	"fe{2}d" finds "feed" but not "fed."
At least n occurrences of the previous character or expression	<code>{n,}</code>	"fe{1,}d" finds "fed" and "feed."
From n to m occurrences of the previous character or expression	<code>{n,m}</code>	"10{1,3}" finds "10," "100," and "1000."
One or more occurrences of the previous character or expression	<code>@</code>	"lo@t" finds "lot" and "loot."
The beginning of a word	<code><</code>	"<(inter)" finds "interesting" and "intercept," but not "splintered."
The end of a word	<code>></code>	"(in)>" finds "in" and "within," but not "interesting."

You can use parentheses around parts of the search criteria to indicate the order of evaluation and to group parts of expressions, as shown in the previous examples.

To search for operators as if they were characters, precede them with a backslash (`\`). For example, to find a question mark, specify `.Find = "\?"` in an `EditFind` instruction.

One other operator that you can specify for `.Replace` in an `EditReplace` instruction is `\num`. This operator rearranges expressions specified in `.Find` in the order specified by `.Replace`. If you specify `.Find = "(Newton) (Christie)"` and `.Replace = "\2 \1"`, the text would change from "Newton Christie" to "Christie Newton."

See also

EditFind

EditReplace

FileFind

Finding and replacing special characters by using keyboard codes

Using special characters in a document search

Finding and replacing special characters by using keyboard codes

To find or replace special characters using an EditFind or EditReplace instruction, specify the following codes for the .Find and .Replace arguments. Press SHIFT+6 for the ^ symbol and make sure to use lowercase letters.

Special characters

To specify	Specify	For
Paragraph mark (¶)	"^p"	.Find or . Replace
Tab character (→)	"^t"	.Find or . Replace
Annotation mark	"^a"	.Find
ANSI or ASCII characters	"^0nnn" where nnn is the character number	.Find or . Replace
Any character	"^?"	.Find
Any digit	"^#"	.Find
Any letter	"^\$"	.Find
Caret character	"^^"	.Find or . Replace
Clipboard contents	"^c"	.Replace
Text specified by . Find	"^&"	.Replace
Endnote mark	"^e"	.Find
Field	"^d"	.Find
Footnote mark	"^f"	.Find
Graphic	"^g"	.Find
Breaks		
To specify	Specify	For
Column break (.....Column Break.....)	"^n"	.Find or . Replace
Line break (↵)	"^l"	.Find or . Replace
Manual page break (.....)	"^m"	.Find or . Replace
Section break (:::::End of Section:::::)	"^b"	.Find
Hyphens and spaces		
To specify	Specify	For
Em dash	"^+"	.Find or .

En dash	"^="	Replace .Find or . Replace
Nonbreaking space (*))	"^s"	.Find or . Replace
Nonbreaking hyphen (-)	"^~"	.Find or . Replace
Optional hyphen (-) }1	"^_"	.Find or . Replace
White space (2)	"^w"	.Find

You cannot search for hyphens that Word inserted automatically with the Hyphenation command (Tools menu).

Note

-) If you omit the optional hyphen code, Word finds all matching text, including text with optional hyphens. If you include the optional hyphen code, Word finds only words with optional hyphens in the same position. For example, if you specify .Find = "type^-writer" Word finds "type-writer", but not "typewriter".
-) Any number and combination of normal and nonbreaking spaces, tab characters, and paragraph marks.

See also

[EditFind](#)

[EditReplace](#)

[Advanced search criteria](#)

Auto Macros

By giving a macro a special name, you can run it automatically when you perform an operation such as starting Word or opening a document. Word recognizes the following names as automatic, or "auto," macros.

Macro name	When it runs
AutoExec	When you start Word
AutoNew	Each time you create a new document
AutoOpen	Each time you open an existing document
AutoClose	Each time you close a document
AutoExit	When you quit Word

Just like other macros, auto macros can be defined either globally or for a particular template. The only exception is the AutoExec macro, which will not run automatically unless it is stored in the Normal template or a global template stored in the directory specified as the Startup directory.

Tip

You can hold down the SHIFT key to prevent auto macros from running. For example, if you create a new document based on a template that contains an AutoNew macro, you can prevent the AutoNew macro from running by holding down SHIFT when you click the OK button in the New dialog box (File menu) and continuing to hold down SHIFT until the new document is displayed. In a macro that might trigger an auto macro, you can use DisableAutoMacros to prevent auto macros from running.

Creating Dynamic Dialog Boxes

To create a dynamic dialog box, you start with a standard dialog box definition created with Begin Dialog..End Dialog. You then add three elements to make the dialog box dynamic:

- A dialog function argument in the Begin Dialog instruction that calls the dialog function. The . FunctionName argument matches the name of the dialog function.
- String identifiers for any dialog box controls that the dialog function acts on or gets information from. Most of the instructions in a custom dialog box definition already include string identifiers for the controls they define.

Note that you can also use numeric identifiers to refer to controls in a dialog box definition (0 (zero) for the first control, 1 for the second control, and so on). Although this may improve performance when a dialog box contains many controls, instructions that use numeric identifiers are more difficult to read than instructions that use string identifiers.

- A dialog function. The dialog function responds to events and changes the appearance of the dialog box. All the instructions that are carried out while the dialog box is displayed are either placed within this function or in subroutines and user-defined functions called from this function.

The following topics describe WordBasic statements and functions used in dialog functions and provide examples of their use. For more information, see Dialog Function Syntax.

DlgControlId()

DlgEnable, DlgEnable()

DlgFilePreview, DlgFilePreview\$()

DlgFocus, DlgFocus\$()

DlgListBoxArray, DlgListBoxArray()

DlgSetPicture

DlgText, DlgText\$()

DlgUpdateFilePreview

DlgValue, DlgValue()

DlgVisible, DlgVisible()

For a complete discussion of creating dynamic dialog boxes, see Chapter 5, "Working with Custom Dialog Boxes," in the Microsoft Word Developer's Kit.

See also

Dialog Function Syntax

Dialog Function Syntax

Function FunctionName(ControlID\$, Action, SuppValue)

Series of instructions

FunctionName =

value

End Function

A dialog function is associated with a dialog box definition when FunctionName matches the . FunctionName argument in a Begin Dialog instruction. By default, the dialog function returns 0 (zero) when the user chooses the OK button, Cancel button, or a push button; a return value of 0 (zero) causes Word to close the dialog box. To keep the dialog box displayed and allow the user to carry out multiple commands from the same dialog box, use the syntax FunctionName = value to set FunctionName to a nonzero value. For an example of this technique, see [DlgText Example](#).

A dialog function takes three required arguments.

Argument	Explanation
ControlID\$	Receives the identifier string of the dialog box control associated with a call to the dialog function. For example, if the user selects a check box, the dialog function is called and the ControlID\$ argument receives the identifier for the check box.
Action	Identifies the action that calls the dialog function. There are six possible actions that can call the dialog function, each with a corresponding Action value. For more information, see the table of Action values, below.
SuppValue	Receives supplemental information about a change in a dialog box control. The information SuppValue receives depends on the Action value and on which control calls the dialog function. For more information, see the table of SuppValues, below.

The following table describes each of the six actions that can call the dialog function.

Action value	Meaning
1	Corresponds to dialog box initialization. This value is passed before the dialog box becomes visible.
2	Corresponds to choosing a command button or changing the value of a dialog box control (with the exception of typing in a text box or combo box). When Action is 2, ControlID\$ corresponds to the identifier for the control that was chosen or changed.
3	Corresponds to a change in a text box or combo box. This value is passed when a control loses the focus (for example, when the user presses the TAB key to move to a different control) or after the user clicks an item in the list of a combo box (an Action value of 2

- is passed first). Note that if the contents of the text box or combo box do not change, an Action value of 3 is not passed. When Action is 3, ControlID\$ corresponds to the identifier for the text box or combo box whose contents were changed.
- 4 Corresponds to a change of control focus. When Action is 4, ControlID\$ corresponds to the identifier of the control that is gaining the focus. SuppValue corresponds to the numeric identifier for the control that lost the focus. A dialog function cannot display a message box or Word dialog box in response to an Action value of 4.
- 5 Corresponds to an idle state. As soon as the dialog box is initialized, Word continuously passes an Action value of 5 while no other action occurs. If the dialog function responds to an Action value of 5, the dialog function should return a nonzero value. (If the dialog function returns 0 (zero), Word continues to send idle messages only when the mouse moves.) When Action is 5, ControlID\$ is an empty string (" ") ; SuppValue corresponds to the number of times an Action value of 5 has been passed so far.
- 6 Corresponds to the user moving the dialog box. This value is passed only when screen updating is turned off (using a ScreenUpdating instruction). After this value is passed and the dialog function ends, Word refreshes the screen and then turns screen updating back on. A dialog function does not usually need to respond to an Action value of 6, but with it you can use the dialog function to change what will be displayed when the screen refreshes. When Action is 6, ControlID\$ is an empty string (" ") ; SuppValue is equal to 0 (zero).

The following table describes which SuppValue values are passed when Action is 2 or 3.

Control	SuppValue passed
List box, drop-down list box, or combo box	Number of the item selected, where 0 (zero) is the first item in the list box, 1 is the second item, and so on
Check box	1 if selected, 0 (zero) if cleared
Option button	Number of the option button selected, where 0 (zero) is the first option button within a group, 1 is

	the second option button, and so on
Text box	Number of characters in the text box
Combo box	If Action is 3, number of characters in the combo box
Command button	A value identifying the button chosen. This value is not often used, since the same information is available from the ControlID\$ value. If the OK button is chosen, SuppValue is 1; if the Cancel button is chosen, SuppValue is 2. The SuppValue for push buttons is an internal number used by Word. This number is not the same as the numeric identifier for a push button, but it does change if the instruction that defines the push button changes position within the dialog box definition.

See also

[Creating Dynamic Dialog Boxes](#)

Converting Word Version 2.x Macros

Overview

WW2_ statements and functions

Working with paragraph marks

Modifying startup options

Error checking

Limits in Word

Creating and displaying dialog boxes

Cutting and pasting text

Using SendKeys

Finding and replacing text

Searching for fields

Working with headers and footers

New formatting implementations

Replacing Windows API calls with new statements and functions

Issues of context when calling macros and subroutines

Taking advantage of global templates

Miscellaneous "gotchas"

Naming variables, subroutines and user-defined functions

Overview

Word converts the macros in a Word 2.x template the first time you open the template, create a new document based on the template, or attach the template to a document using the Templates command (File menu). (Note that Word 6.0 cannot convert Word 1.x macros directly; open Word 1.x templates first in Word 2.x, and then in Word 6.0.) After converting a template, you must save it to save the conversion. If you don't save the template, Word converts the macros again the next time you use the template.

If you want complete control over converting macros (that is, if you don't want Word to automatically convert your macros), you can convert your macros manually. To do so, open each macro in Word 2.x, copy the code to a normal document, and save the document. In Word 6.0, open the document and in either the Normal template or a new custom template, create a macro for each of your original macros, then copy the text from the document into the macro editing window. Debug each macro to identify which parts of your code should be changed to produce the same behavior you programmed in Word 2.x.

When Word converts your macros automatically, you may need to modify parts of them by hand to complete the conversion. This topic attempts to identify areas of your macro to which you may need to pay special attention to produce the behavior you want from your macro.

WW2_ statements and functions

For improved compatibility, a number of Word 2.x WordBasic statements and functions have been carried over to Word 6.0 and given the "WW2_" prefix (for example, WW2_CountMenuItems() and WW2_EditFind). When the macro converter encounters one of these Word 2.x statements or functions, it substitutes the WW2_ name.

WW2_ functions provide Word 2.x syntax, but they do not behave under Word 2.x assumptions. For example, WW2_Insert adheres to the setting of the Smart Cut And Paste setting in Word 6.0, and WW2_EditFind must use Word 6 codes to search for special characters.

Here is the full list of WW2_ statements and functions:

WW2_ChangeCase	WW2_MenuText\$()
WW2_ChangeRulerMode	WW2_PrintMerge
WW2_CountMenuItems()	WW2_PrintMergeCheck
WW2_EditFind	WW2_PrintMergeCreateDataSource
WW2_EditFindChar	WW2_PrintMergeCreateHeaderSource
WW2_EditReplace	WW2_PrintMergeHelper
WW2_EditReplaceChar	WW2_PrintMergeSelection
WW2_FileFind	WW2_PrintMergeToDoc
WW2_FileTemplates	WW2_PrintMergeToPrinter
WW2_Files\$()	WW2_RenameMenu
WW2_FootnoteOptions	WW2_RulerMode
WW2_FormatBordersAndShading	WW2_TableColumnWidth
WW2_FormatCharacter	WW2_TableRowHeight
WW2_FormatDefineStyleChar	WW2_ToolsHyphenation
WW2_GetToolButton()	WW2_ToolsMacro
WW2_GetToolMacro\$()	WW2_ToolsOptionsGeneral
WW2_Insert	WW2_ToolsOptionsKeyboard
WW2_InsertFootnote	WW2_ToolsOptionsMenus
WW2_InsertIndex	WW2_ToolsOptionsPrint
WW2_InsertSymbol	WW2_ToolsOptionsToolbar
WW2_InsertTableOfContents	WW2_ToolsOptionsView
WW2_KeyCode	WW2_ToolsRevisionsMark
WW2_MenuMacro\$()	WW2_ViewZoom

Note that CommandValid() takes a string that specifies a command name. Word 6 does not convert this string to a valid Word 6 command name, nor does it append "WW2_." Check all occurrences of this function in a converted macro to ensure the name of the command being tested is valid (for example, change "InsertBookmark" to "WW2_InsertBookmark" or "EditBookmark").

Keep in mind the following details about the behavior of some Word 2.x and WW2_ commands compared to the corresponding Word 6.0 commands.

- WW2_Files\$() returns the filename only, while the Word 6.0 Files\$() function returns the path and filename.
- WW2_Insert has the same effect as the Word 6.0 Insert statement except when text is selected. If the current selection includes a section break at the end of the selection, WW2_Insert overwrites it; the Word 6.0 Insert statement does not. If a word is selected, including the space character following it, WW2_Insert replaces the trailing space character; the Word 6.0 Insert statement does not.
- Word 6.0 provides compatibility in find and replace macro operations by including the Word 2.x versions of these statements as WW2_EditFind and WW2_EditReplace. Note that the Word 2.x special character codes continue to work in WW2_EditFind and WW2_EditReplace. However, specifying ANSI character 34 (straight quotation mark) as the find text in WW2_EditFind, WW2_EditReplace, EditFind, or EditReplace in Word 6.0 finds both straight and "smart" quotation marks (ANSI 147 and 148); in Word 2.x macros, ANSI 34 finds only straight quotation marks.
For more information on changes to finding and replacing, see [Finding and replacing text](#).
- The Word 2.x statement ViewHeaderFooter is supported in Word 6.0 as the NormalViewHeaderArea statement; however, you cannot display the Word 2.x dialog box with NormalViewHeaderArea.
- The Word 2.x statement IconBarMode is supported in Word 6.0, but has no effect.

Some WW2_ statements correspond to dialog boxes in Word 2.x. A subset of these statements cannot be used to display the Word 2.x dialog boxes in Word 6.0 (though the statements may still be used to set options or return information through dialog records). Converted Word 2.x macros that attempt to display a dialog box associated with any of the following statements will need to be updated by hand.

NormalViewHeaderArea
WW2_EditFindChar

WW2_EditReplaceChar
WW2_FormatDefineStyleChar
WW2_PrintMerge
WW2_PrintMergeCheck
WW2_PrintMergeHelper
WW2_PrintMergeSelection
WW2_PrintMergeToDoc
WW2_PrintMergeToPrinter
WW2_ToolsOptionsGeneral
WW2_ToolsOptionsKeyboard
WW2_ToolsOptionsMenus
WW2_ToolsOptionsToolbar
WW2_ToolsOptionsView
WW2_ViewZoom

Working with paragraph marks

In Word 2.x, the two ANSI characters 13 and 10 specified a paragraph mark. In Word 6.0, the single ANSI character 13 represents a paragraph mark. Any Word 2.x macro that assumes the following:

```
para$ = Chr$(13) + Chr$(10)
```

will not work properly in a Word 6.0 document. Word 2.x macros often use this assumption to search for paragraph marks or to test a selection to see if it contains a paragraph mark. Changing this assumption in any converted Word 2.x macro will remedy this incompatibility with Word 6.0 documents.

However, paragraph marks in Word 2.x and text-only documents opened in Word 6.0 are still equivalent to ANSI characters 13 and 10; only when a Word 2.x or text-only document is finally saved in Word 6.0 format are the paragraph marks converted to ANSI character 13. If your macro needs to work on documents in both formats, make sure to check the current format before setting the assumption for which ANSI character or characters comprise a paragraph mark.

Modifying startup options

Startup options for Word 6.0 are now in WINWORD6.INI. Macros that specify a Word section ("Microsoft Word 2.0," "Microsoft Word," "MSWord Text Converters," or "MSWord Editable Sections") in a GetProfileString\$() or SetProfileString instruction will return or set information in WINWORD6.INI instead of WIN.INI. If you need to return or set options in Word 2.x sections of WIN.INI, use GetPrivateProfileString\$() and SetPrivateProfileString, which allow you to explicitly specify an INI file.

Error checking

Because error messages in Word 6.0 are more specific than those in Word 2.x, you may need to update error-handling routines to trap new errors. For example, if the insertion point or selection is not in a table, the StartOfRow and EndOfRow statements will generate an error message. Also, keep the following points in mind:

- Routines that manipulate dialog boxes without using GetCurValues may generate errors in Word 6.0 that did not occur in Word 2.x.
- Word 6.0 now displays an error if an array variable specified in a dialog box definition has not been defined.

Limits in Word

You may want to fix assumptions your macros make about Word limits that have changed (for example, the maximum number of open document windows has changed from nine to whatever number available memory allows). A change to consider when converting Word 2.x macros is that the number of nesting levels for Call instructions to other macros and subroutines has been reduced. But as in Word 2.x, available memory often limits the number of nesting levels before a macro can reach the internal maximum, around 9 in Word 6.0.

For more information on new limits and other changes in Word 6.0, see What's New in WordBasic and Chapter 6, "Switching from a Previous Version of Word," in Microsoft Word Quick Results.

Creating and displaying dialog boxes

In Word 2.x, option buttons and check boxes are vertically centered within the rectangle defined by the width and height arguments in `OptionButton` and `CheckBox` instructions. In Word 6.0, option buttons and check boxes are aligned at the top of the rectangle. If the rectangle was larger than necessary in the Word 2.x macro, the option button or check box may be out of place when the dialog box definition is converted. If necessary, paste the dialog box definition into the Dialog Editor and resize the controls.

In Word 6.0, list boxes no longer recognize empty strings. If a macro includes an empty string in an array to be assigned to a list box, the list of entries is truncated after the empty string. For example, if you create the following array:

```
ListBox1$(0) = "hello"  
ListBox1$(1) = ""  
ListBox1$(2) = "hello"
```

and then assign it to a list box in a dialog box definition, no text will appear in the list box after "hello" when the dialog box is displayed. To fix the dialog box, either eliminate the empty string from the array or add a space to each empty string. For example:

```
ListBox1$(1) = " "
```

In Word 6.0, a custom dialog box with no Cancel button cannot be closed using the dialog Control menu. Two approaches can be taken to address this: Use the `MsgBox` command instead of a custom dialog box (note that a message box can only display 256 characters), or include a Cancel button to the dialog box definition and then create a dialog box function that hides the Cancel button on initialization.

In Word 2.x, input boxes displayed with `InputBox$()` set the focus on the OK button; to choose OK using the keyboard, the user pressed `ENTER`, and to insert a new line break in the text box, the user pressed `SHIFT+ENTER`. In Word 6.0, input boxes displayed with `InputBox$()` set the focus on the text box. When the user presses `ENTER`, Word inserts a new line in the text box; to choose OK using the keyboard, the user must press `TAB` to set the focus on the OK button and then press `ENTER`.

If you want to maintain the Word 2.x `InputBox$()` behavior in your macro, you need to create a custom dialog box to display with a `Dialog` or `Dialog()` instruction instead of using `InputBox$()`. If you do use the Word 6.0 `InputBox$()` function, you can make your macro more robust by evaluating the returned string to ensure that it is usable in your macro, cleaning it up if the user inadvertently pressed `ENTER` while trying to choose OK.

Cutting and pasting text

The Edit panel in the Options dialog box (Tools menu) contains a new editing option, Use Smart Cut And Paste, that removes unneeded spaces when you delete text and adds spaces when you insert text. In macros that delete, cut, or paste text, use ToolsOptionsEdit to control this option, making sure the setting of the option corresponds to your macro's assumptions. For Word 2.x macros, the assumption will most likely be that this feature is not available, so add the following instructions to your macro to make sure it behaves the same in Word 6.0:

```
Sub MAIN
Dim dlg As ToolsOptionsEdit
GetCurValues dlg
reset = dlg.SmartCutPaste
dlg.SmartCutPaste = 0
ToolsOptionsEdit dlg
' Word 2.x macro instructions
ToolsOptionsEdit .SmartCutPaste = reset
End Sub
```

Using SendKeys

The macro converter does not change the keystrokes specified in SendKeys statements to accommodate changes to access keys for menus, menu items, and dialog box controls in Word 6.0. For example, in Word 2.x, the instruction

```
SendKeys "%ob"
```

displays the Bullets And Numbering dialog box (Tools menu). In Word 6.0, the same instruction displays the Borders And Shading dialog box (Format menu). Search your converted Word 2.x macros for all SendKeys instructions to verify that they will still function as expected in Word 6.0.

Finding and replacing text

The EditFind and EditReplace statements have been updated for Word 6.0. The new versions use different values for .Direction and use the new .Wrap argument to control prompts (for details, see EditFind). Also, a few of the character codes used when searching for and replacing special characters have changed (for example, "^m" instead of "^d" for a manual page break).

For these reasons, Word 6.0 provides compatibility in find and replace macro operations by including the Word 2.x versions of these statements as WW2_EditFind and WW2_EditReplace. Note that the Word 2.x special character codes continue to work in WW2_EditFind and WW2_EditReplace.

In a Word 2.x EditFind instruction, you set .Direction to 2 to search toward the end of the document and prevent Word from displaying a prompt if the end of the document is reached. If there is a selection when the search begins, Word 2.x searches the selection first, and then continues the search after the selection. A WW2_EditFind instruction in Word 6.0 does not continue the search after the selection. Unless your macro makes sure that there is no selection before the WW2_EditFind instruction is run, you may want to rewrite the instruction using the Word 6.0 version of EditFind, setting .Direction to 0 (zero) and the new .Wrap argument to 1.

The font name and ANSI code of symbols inserted using the Word 6.0 InsertSymbol command are hidden; Word recognizes these symbols as ANSI character 40 (left parenthesis). Be aware that converted Word 2.x macros that search for left parentheses will also find symbols inserted with InsertSymbol in Word 6.0 documents.

Searching for fields

In Word 2.x, fields are inserted with no space between the opening field character and the field name. In Word 6.0, a space is inserted after the opening field character and before the closing field character. If you have macros that search for specific fields and perform some action on them, you'll need to take this into account. Consider the following macro converted from Word 2.x. Notice that in the find text, there is no space between ^19 and DATE.

```
REM UnlinkDateFields -- unlinks each DATE field in the document
Sub MAIN
StartOfDocument
EditFindClearFormatting
WW2_EditFind .Find = "^19DATE", .Direction = 2, .Format = 0, .MatchCase = 0
While EditFindFound()
    UnlinkFields
    WW2_EditFind .Find = "^19DATE", .Direction = 2, .Format = 0, .MatchCase = 0
Wend
End Sub
```

You should assume that documents contain fields with varying numbers of spaces after the opening field character, especially if the document began as a Word 2.x document. To account for this, the macro above could be rewritten to include two loops: one for DATE fields with no space after the opening field character and one for DATE fields with one or more spaces (^w) after the opening field character. Note that the following macro uses the Word 6.0 versions of EditFind and EditReplace, in which you can specify ^d for a field character.

```
REM UnlinkDateFields -- unlinks each DATE field in the document
Sub MAIN
EditFindClearFormatting
ViewFieldCodes 1
EditFind .Find = "^dDATE", .Direction = 0, .Wrap = 1, .Format = 0, \
    .MatchCase = 0
While EditFindFound()
    UnlinkFields
    EditFind .Find = "^dDATE", .Direction = 0, .Wrap = 1
Wend
EditFind .Find = "^d^wDATE", .Direction = 0, .Wrap = 1
While EditFindFound()
    UnlinkFields
    EditFind .Find = "^d^wDATE", .Direction = 0, .Wrap = 1
Wend
End Sub
```

You should also be aware that there are four fields whose names have changed in Word 6.0. However, when you open a Word 2.x document containing one or more of these fields in Word 6.0, Word does not update the field names. The fields continue to work as before, but their names don't change to the Word 6.0 names unless you edit the field codes. The following table lists these four fields.

Word 2.x	Word 6.0
INCLUDE	INCLUDETEXT
IMPORT	INCLUDEPICTURE
FTNREF	NOTEREF
GLOSSARY	AUTOTEXT

If you have a macro that searches for one of these fields, you may want to add code that accounts for the possibility that both field names appear in the same document.

Working with headers and footers

In Word 6.0, the most common way to work with headers and footers is to display them with the ViewHeader statement in page layout view. One limitation of this method is that Word can only display the headers and footers of pages that exist in the document (that is, pages that can be displayed in page layout view).

To work with headers and footers for documents with little or no text (for example, a template on which much longer book-like documents will be based), you should use the NormalViewHeaderArea statement to display any header or footer in the header/footer pane in normal view. The NormalViewHeaderArea statement corresponds to the Word 2.x ViewHeaderFooter statement. The arguments are the same, and you can use a dialog record and GetCurValues to return the current values of NormalViewHeaderArea; however, in Word 6.0, you cannot display the Word 2.x ViewHeaderFooter dialog box.

The following Word 6.0 macro instructions are equivalent:

```
FilePageSetup .DifferentFirstPage = 1, .OddAndEvenPages = 1
NormalViewHeaderArea .FirstPage = 1, .OddAndEvenPages = 1
```

But in a document with no text or page breaks, the following instruction displays the odd header in the header/footer pane in normal view:

```
NormalViewHeaderArea .Type = 4
```

while the following instruction can only display the first-page header in page layout view:

```
ViewHeader
```

When enough text is added to create two page breaks (or if page breaks are added using InsertBreak), a ViewHeader instruction would be able to display the odd header in page layout view.

Macros converted from Word 2.x will automatically use the NormalViewHeaderArea statement, just as they used ViewHeaderFooter before. To duplicate Word 2.x functionality in new Word 6.0 macros, you should use NormalViewHeaderArea as well. If you want your Word 6.0 macro to use ViewHeader in page layout view, regardless of the number of pages in the active document or template, write code to insert one or two temporary page breaks, modify the headers and footers, and then remove the temporary page breaks.

New formatting implementations

Word 6.0 has many new formatting features and has revised some Word 2.x formatting statements and functions for greater usability. When converting Word 2.x macros, you may need to rewrite some code that applies formatting if you want to duplicate Word 2.x formatting behavior. Here are some specific situations you might look out for.

- Word 6.0 now has Superscript, Subscript, and Small Caps formats based on the typographical information stored in the specified font. Word 2.x macros that created superscript and subscript text by raising text and reducing its font size manually are converted to do the same in Word 6.0. However, instructions that use this technique can be modified manually to take advantage of the new font formatting capabilities of Word.

- The .LineSpacing argument of the Word 2.x FormatParagraph statement has been split into two arguments in the Word 6.0 FormatParagraph statement: .LineSpacingRule and .LineSpacing. To specify exact line spacing in Word 2.x, you would specify a negative value for the .LineSpacing argument (for example, "-10 pt"). To apply the same formatting in Word 6.0, you can do one of two things: specify Exactly (value 4) for .LineSpacingRule and a positive value for .LineSpacing (for example, "10 pt"); or specify a negative value for the .LineSpacing argument (for example, "-10 pt"). In this way, Word 2.x instructions that apply this formatting are converted without error.

However, after Word 6.0 runs an instruction that assigns .LineSpacing a negative value, the value of .LineSpacingRule is set to 4 ("Exactly") and the value of .LineSpacing is changed to a positive value. Therefore, if your macro contains any conditional statements (such as If...Then...Else or While...Wend) that test for a negative .LineSpacing value in a Word 6.0 FormatParagraph dialog record, they will always return false. Each conditional statement that tests for a negative .LineSpacing value should be modified to test for either a positive .LineSpacing value, a .LineSpacingRule value of 4, or both, depending on the information required.

- Word 6.0 provides two kinds of style: paragraph and character. The name of the current style, returned by the StyleName\$() function, depends on where the insertion point or selection is located. For example, if a word is selected in a Normal paragraph and no character styles are applied to the word, StyleName\$() returns "Normal." However, if the word has a character style, such as ArialBold, applied to it, StyleName\$() returns "ArialBold." To make sure StyleName\$() returns the underlying paragraph style, regardless of the any character styles applied to the current selection, use the following code:

```
EditBookmark "tmp"  
SelType 1  
reset$ = StyleName$()  
Style "Default Paragraph Font"  
parastyle$ = StyleName$()  
Style reset$  
EditGoto "tmp"  
EditBookmark "tmp", .Delete
```

- In Word 6.0, the Organizer command can be used in macros to copy multiple styles, AutoText entries, toolbars, and macros; a macro simply establishes a loop based on the number of items counted by a function such as CountStyles() and runs an Organizer instruction for each item.

As in Word 2.x, macros in Word 6.0 can use the FormatStyle statement to merge all styles to or from documents or templates using the .FileName, .Source, and .Merge arguments. Word 2.x macros that use this method are converted with little or no modification into Word 6.0.

Replacing Windows API calls with new statements and functions

Some new WordBasic statements and functions provide the functionality of common Windows API calls used in Word 2.x with Declare statements. When converting a macro from Word 2.x to Word 6.0, you might consider which Windows API calls the macro made before could be converted to new built-in WordBasic functionality.

For example, the application control statements such as AppSize, AppMove, and AppMinimize can be used in Word 6.0 to control the state of any Windows-based application, not just Word. If your macro attempts to modify the state of non-Word applications using Windows API calls, you might consider replacing the API calls with the corresponding WordBasic statements. Also, new statements such as AppGetNames, AppCount(), and AppIsRunning() extend the ability of macros to modify or return information about the entire Windows environment.

AppSendMessage is a powerful statement added to Word 6.0 that allows macros to send any Windows API message and its associated parameters (described in the Microsoft Windows 3.1 Software Development Kit) to any running Windows-based application. If you are converting a Word 2.x macro that attempted to do the same thing using Windows API function calls, you can modify the macro to take advantage of AppSendMessage.

Word 6.0 has added two statements, ScreenUpdating and ScreenRefresh, to provide some display control that could only be found in calls to the Windows API EchoOff function. Note that ScreenUpdating does not provide the same functionality as EchoOff; toolbars can be hidden and displayed, the status bar can be updated, message boxes can prompt for information, and so on. If you have a Word 2.x macro that used the Windows API EchoOff function, you might consider using the Word 6.0 screen updating statements instead if they satisfy the needs of the macro.

In Word 6.0, you can use GetPrivateProfileString\$() and SetPrivateProfileString to return and modify settings in any initialization file: WIN.INI, WINWORD6.INI, an initialization file for any other Windows-based application, or even your own initialization file such as MACROVAR.INI. If you are converting a Word 2.x macro that uses Windows API calls to functions of the same name, you might consider whether the built-in statement and function can be used to accomplish the same task.

Issues of context when calling macros and subroutines

In both Word 2.x and Word 6.0, you can call one macro from another by using a ToolsMacro instruction or by using the following syntax:

```
[Call] MacroName[.SubName] [ArgumentList]
```

Occasionally, more than one macro with the specified name are available to run. In such cases, Word 6.0 uses different rules than Word 2.x when deciding which macro to run. In general, Word 2.x resolves name conflicts in favor of the active template and Word 6.0 resolves name conflicts in favor of the template that contains the calling macro. An example will illustrate this point.

Consider the template MY.DOT containing the macro Welcome.

```
'Welcome macro (MY.DOT)
Sub Main
MsgBox "I am the Welcome macro in MY.DOT"
End Sub
```

Consider a macro of the same name in NORMAL.DOT.

```
'Welcome macro (NORMAL.DOT)
Sub Main
MsgBox "I am the Welcome macro in NORMAL.DOT"
End Sub
```

Now consider another macro in the Normal template which creates a document based on MY.DOT, and then runs Welcome.

```
'Macro in NORMAL.DOT that runs NORMAL.DOT version of Welcome
Sub Main
FileNew .Template = "MY.DOT"
Welcome
End Sub
```

When the Welcome macro runs, MY.DOT is active. In Word 2.x, the MY.DOT version of Welcome runs because naming conflicts are resolved in favor of the active template. In Word 6.0, where the version in the template containing the calling macro takes precedence over the version in the active template, the NORMAL.DOT version of Welcome runs.

How can you override this behavior in Word 6.0 macros without renaming all of your macros and subroutines to use unique names? There are two ways: use ToolsMacro instead of Call, or include the WW2CallingConvention statement.

If you want to run the main subroutine of a macro in the active template regardless of which template contains the calling macro (and you don't need to pass any values), use a ToolsMacro instruction and set .Show to 3 (the value for active template context).

```
'Macro in NORMAL.DOT that runs MY.DOT version of Welcome
Sub Main
FileNew .Template = "MY.DOT"
ToolsMacro .Name = "Welcome", .Show = 3, .Run
End Sub
```

Note that whenever you call a macro with ToolsMacro, it's a good idea to specify .Show. Otherwise, the context will be determined by whatever context was last selected in the Macro dialog box. This is different from the Word 2.x version of ToolsMacro, where, if you omitted .Show, Word looked for the macro first in the active template, then in the Normal template, and finally in built-in commands.

Use WW2CallingConvention if you want Word 6.0a to resolve naming conflicts the same way Word 2.x resolved them. With WW2CallingConvention, your macros can use Call to call macros or subroutines within macros in an active template if a macro by the same name already exists in the calling template. You must include WW2CallingConvention if you want to pass values to a macro with a conflicting name that you might otherwise call with a ToolsMacro statement that has .Show set to 3, as described above.

Note

The WW2CallingConvention statement is an addition to Word 6.0a. Note that anyone using a macro that contains WW2CallingConvention must also have Word 6.0a for the macro to perform as intended. Including a CommandValid() check in your macros that include WW2CallingConvention will prevent a user from trying to run the new statement when it is not available in their version of Word. For more information, see [WW2CallingConvention](#).

Taking advantage of global templates

If you are converting a complex suite of macros in multiple templates from Word 2.x to Word 6.0, you should consider taking advantage of global templates in Word 6.0 for the following reasons:

- In Word 2.x, it was common practice to distribute macros in a template that ran a process using MacroCopy to copy some or all of the macros to a user's Normal template so those macros would be available at all times. In Word 6.0, you need only distribute a template containing all of your macros and instruct the user to load the template as a global template, using the Templates And Add-ins dialog box (File menu). With global templates, you don't have to touch your user's Normal template.
- If you distributed multiple Word 2.x templates, each with its own set of macros, you can reorganize those templates to take advantage of global templates. The macros for manipulating a new document based on any given template do not need to be stored in the specific template; rather, they can all reside in one authoritative global template. You can also avoid cross-template naming conflicts by storing macros in one global template.
- The change from juggling templates in Word 2.x to loading a single global template in Word 6.0 to automatically customize a user's Word environment requires some recoding and reorganizing of existing Word 2.x template suites. However, the global-template model for customizing Word will pay off by giving converted Word 2.x templates long-term stability in Word 6.0 and later versions.

Miscellaneous "gotchas"

Look for the following assumptions in your Word 2.x macros when you convert them to Word 6.0. A change in Word 6.0 behavior may cause your macro to behave unexpectedly or incorrectly if it operates under one of these conditions.

- The predefined bookmarks "\Para" and "\Page" no longer select the last paragraph mark in a document if that paragraph mark is adjacent to the rest of the bookmark. For example, in Word 2.x, an EditGoto instruction that specified "\Page" would select the last paragraph mark in the document if the insertion point or selection was in the last page; in Word 6.0, the paragraph mark is excluded. You need to modify a converted Word 2.x macro if it continues after such an instruction with the assumption that the paragraph mark is part of the selection.

- In Word 2.x, if your macro used FileSaveAs to save the active document in a file format other than Word Document, Word saved the new version of the file but left the original active document active. In Word 6.0, Word saves the new version of the file, closes the original active document (if it had already been saved), and makes the new version of the file in the foreign format the active document.

If your converted Word 2.x macro assumes that any editing done after saving a file in a foreign format is being done on the original Word Document file, it will behave incorrectly; it will actually modify the content of the foreign-format file, which is active. Modify your Word 2.x macro in Word 6.0 to close the foreign-format file and re-open the original Word Document file if it needs to continue modifying the Word Document version of the file.

- A macro that includes an OnTime instruction will run the specified macro regardless of whether Word is the active application when the specified time occurs. Word 2.x, if the specified time passed while Word was inactive, Word ran the macro as soon as it became the active application. Any Word 2.x macro that assumed Word would be the active application when the specified macro ran as a result of OnTime should be modified to work with the new assumption or use another kind of delay routine.

Naming variables, subroutines and user-defined functions

You may need to change names of variables, subroutines, and user-defined functions if the names have become reserved words in Word 6.0 (such as statement or function names).

You cannot call a subroutine or user-defined function stored in another macro if the name of the subroutine or function is the same as the name of an argument for a WordBasic statement that corresponds to a dialog box. For example, if you have a macro called "Library" that contains a subroutine called "Wrap," you cannot call the subroutine from another macro in the same template. The instruction

```
Library.Wrap
```

in another macro generates an error because .Wrap is an argument of the EditFind statement.

If you locate a Word 2.x macro that contains a subroutine or user-defined function with a name that has become a reserved word, you should change the name of the subroutine or function to avoid the error described above.

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

ClearFormField Example

This example is intended to run when the focus moves to a text form field. If the user moves to the form field using the TAB key, thereby selecting its contents, the condition `GetSelStartPos() <> GetSelEndPos()` is true and Word clears the form field. If the user clicks the form field with the mouse, the condition is false and Word takes no action.

```
If GetSelEndPos() <> GetSelStartPos() Then ClearFormField
```

ClearFormField

ClearFormField

Clear the text in a text form field selected in a protected form document. ClearFormField behaves like the BACKSPACE key. Note that in an unprotected form document, ClearFormField deletes the selected text form field (unless the form field was selected while the document was protected, in which case the form field's text is cleared, and the form field is not deleted). An error occurs if a text form field is not selected; the statement cannot be used to clear a drop-down or check box form field.

See also

Forms Statements and Functions

SetFormResult

TextFormField

ewc shareres, T3EWClass, \$\$button:WordBASICCclosewc shareres, T3EWClass, \$\$button:WordBASICCcopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

ConvertObject Example

This example changes the display of the selected embedded object to an icon stored in PROGMAN.EXE.

```
ConvertObject .IconNumber = 28, .IconFilename = "PROGMAN.EXE", \  
  .Caption = "Caption Text", .DisplayIcon = 1
```

ConvertObject

Example

ConvertObject [.IconNumber = number] [, .ActivateAs = number] [, .IconFilename = text] [, .Caption = text] [, .Class = text] [, .DisplayIcon = number]

Converts the selected embedded object from one class to another, allows a different server application to edit the object, or changes how the object is displayed in the document. The arguments for the ConvertObject statement correspond to the options in the Convert dialog box (Object submenu, Edit menu).

Argument	Explanation
.IconNumber	If .DisplayIcon is set to 1, a number corresponding to the icon you want to use in the program file specified by .IconFilename. Icons appear in the Change Icon dialog box (Object command, Insert menu): 0 (zero) corresponds to the first icon, 1 to the second icon, and so on. If omitted, the first (default) icon is used.
.ActivateAs	Specifies whether Word converts or sets the server application for the selected object: 0 (zero) Converts the selected object to the object type specified by .Class. 1 Uses the server application specified by .Class to edit the object. Note that this setting applies to all objects of the selected type and that Word uses the specified server application when inserting objects of the selected type.
.IconFilename	If .DisplayIcon is set to 1, the path and filename of the program file in which the icon to be displayed is stored.
.Caption	If .DisplayIcon is set to 1, the caption of the icon to be displayed; if omitted, Word inserts the name of the object.
.Class	A class name specifying the object type to convert to or the server application for editing the object, depending on the setting for .ActivateAs. The class name for a Word document is Word.Document.6 and a Word picture is Word.Picture.6. To look up other class names, insert an object of the type to convert to in a document and view the field codes; the class name of the object follows the word "EMBED."
.DisplayIcon	Specifies whether or not to display the object as an icon: 0 (zero) or omitted Object is not displayed as an icon. 1 Object is displayed as an icon.

See also

[Object Linking and Embedding Statements and Functions](#)

[InsertObject](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCclosewc shareres, T3EWClass, \$\$button:WordBASICCcopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

CountDocumentVars() Example

This example resets each document variable in the active document to an empty string (" "). If the document contains no variables, a message box is displayed.

```
numVars = CountDocumentVars()  
If numVars > 0 Then  
    For i = 1 To CountDocumentVars()  
        name$ = GetDocumentVarName$(i)  
        SetDocumentVar name$, "  
    Next  
Else  
    MsgBox "No document variables to reset."  
End If
```

CountDocumentVars()
CountDocumentVars()

~~Example~~ Returns the number of document variables set with SetDocumentVar or SetDocumentVar() in the active document.

See also

Documents, Templates, and AddIns Statements and Functions

GetDocumentVar\$()

GetDocumentVarName\$()

SetDocumentVar

DrawResetWordPicture

Resets the boundaries in a Word Picture object to include all drawing objects in the picture editing window. If the active window is not a picture editing window, an error occurs.

See also

Drawing Statements and Functions

DrawInsertWordPicture

FileClosePicture

FieldSeparator\$, FieldSeparator\$()

FieldSeparator\$ Separator\$

FieldSeparator\$()

The FieldSeparator\$ statement sets the separator character, Separator\$, Word recognizes when dividing text among cells in a TextToTable operation. For example, if you have data in which the items of information are delimited by percent signs (%), you can use the instruction `FieldSeparator$ "%"` before converting the data to a table. The FieldSeparator\$() function returns the current separator character.

See also

Tables Statements and Functions

TextToTable

FileClosePicture

Closes the picture editing window and embeds a Word Picture object in the document.

See also

[Drawing Statements and Functions](#)

[DrawResetWordPicture](#)

FormatBullet

FormatBullet [.Points = number] [, .Color = number] [, .Alignment = number] [, .Indent = number or text] [, .Space = number or text] [, .Hang = number] [, .CharNum = number] [, .Font = text]

Adds bullets to the selected paragraphs. The arguments for the FormatBullet statement correspond to the options in the Modify Bulleted List dialog box (Bulleted tab, Bullets And Numbering command, Format menu). You cannot display this dialog box using a Dialog or Dialog() instruction.

Argument	Explanation
.Points	The size of the bullets, in points.
.Color	The color of the bullets (for a list of colors, see CharColor).
.Alignment	Specifies an alignment for the bullets within the space between the left indent and the first line of text; takes effect only if .Space is 0 (zero): 0 (zero) or omitted Left 1 Centered 2 Right
.Indent	The distance between the left indent and the first line of text, in points or a text measurement.
.Space	The distance between the bullet and the first line of text, in points or a text measurement.
.Hang	If 1, applies a hanging indent to the selected paragraphs.
.CharNum	The sum of 31 and the number corresponding to the position of the symbol in the Symbol dialog box (Insert menu), counting from left to right. For example, to specify an omega (ω), which is at position 56 on the table of symbols in the Symbol font, set .CharNum to 87.
.Font	The name of the font containing the symbol. Names of decorative fonts appear in the Font box in the Symbol dialog box.

See also

[Bullets and Numbering Statements and Functions](#)

[CharColor](#)

[FormatBulletsAndNumbering](#)

[FormatHeadingNumber](#)

[FormatMultilevel](#)

[FormatNumber](#)

FormatHeadingNumber

FormatHeadingNumber [.Points = number] [, .Color = number] [, .Before = text] [, .Type = number] [, .After = text] [, .StartAt = number] [, .Include = number] [, .Alignment = number] [, .Indent = number or text] [, .Space = number or text] [, .Hang = number] [, .RestartNum = number] [, .Level = number] [, .Font = text] [, .Strikethrough = number] [, .Bold = number] [, .Italic = number] [, .Underline = number]

Applies numbers to all paragraphs in the document formatted with one of the nine built-in heading level styles, or changes numbering options for a specified heading level. The arguments for the FormatHeadingNumber statement correspond to the options in the Modify Heading Numbering dialog box (Heading Numbering command, Format menu).

Argument	Explanation
.Points, .Color, .Font, .Strikethrough, .Bold, .Italic, .Underline	Apply character formatting to numbers at the specified level. For argument descriptions, see FormatFont .
.Before, .After, .Alignment, .Indent, .Space, .Hang	Set options for numbers at the specified level. For argument descriptions, see FormatNumber .
.Type	Specifies a format for numbering headings at the specified level: 0 (zero) 1, 2, 3, 4 1 I, II, III, IV 2 i, ii, iii, iv 3 A, B, C, D 4 a, b, c, d 5 1st, 2nd, ... 6 One, Two, ... 7 First, Second, ...
.StartAt	The number for the first heading in each sequence of headings of the specified level. If .Type is 3 or 4, .StartAt corresponds to the position in the alphabet of the starting letter.
.Include	Specifies whether to include numbers and position options from the previous headings for numbers at the specified level: 0 (zero) Includes neither numbers nor position options. 1 Includes a series of numbers from higher-level headings before the numbers at the specified level. 2 Includes both numbers from higher-level headings and position options from the previous level.
.RestartNum	If 1, restarts heading numbering at each new section.
.Level	A number from 1 through 9 corresponding to the heading level whose numbering options you want to change.

See also

[Bullets and Numbering Statements and Functions](#)

[FormatBullet](#)

[FormatHeadingNumbering](#)

[FormatMultilevel](#)

[FormatNumber](#)

FormatMultilevel

FormatMultilevel [.Points = number] [, .Color = number] [, .Before = text] [, .Type = number] [, .After = text] [, .StartAt = number] [, .Include = number] [, .Alignment = number] [, .Indent = number or text] [, .Space = number or text] [, .Hang = number] [, .Level = number] [, .Font = text] [, .Strikethrough = number] [, .Bold = number] [, .Italic = number] [, .Underline = number]

Applies multilevel list numbers to the selected paragraphs or changes numbering options for a specified level. The arguments for the FormatMultilevel statement correspond to the options in the Modify Multilevel List dialog box (Multilevel tab, Bullets And Numbering command, Format menu). You cannot display this dialog box using a Dialog or Dialog() instruction.

Argument	Explanation
.Level	A number from 1 through 9 corresponding to the heading level whose numbering options you want to change. Note that if you specify .Level, the options you set in the FormatMultilevel instruction are not applied. To apply the settings, include a second FormatMultilevel instruction in which .Level is not specified.
.Points, .Color, .Font, .Strikethrough, .Bold, .Italic, .Underline	Apply character formatting to numbers at the specified level. For individual argument descriptions, see FormatFont .
.Before, .After, .Alignment, .Indent, .Space, .Hang	Set options for numbers at the specified level. For argument descriptions, see FormatNumber .
.Type	Specifies a format for numbering headings at the specified level: 0 (zero) 1, 2, 3, 4 1 I, II, III, IV 2 i, ii, iii, iv 3 A, B, C, D 4 a, b, c, d 5 1st, 2nd, ... 6 One, Two, ... 7 First, Second, ...
.StartAt	The number for the first heading in each sequence of headings of the specified level. If .Type is 3 or 4, .StartAt corresponds to the position in the alphabet of the starting letter.
.Include	Specifies whether to include numbers and position options from the previous headings for numbers at the specified level: 0 (zero) Includes neither numbers nor position options. 1 Includes a series of numbers from higher-level headings before the numbers at the specified level. 2 Includes both numbers from higher level-headings and position options from the previous level.

See also

[Bullets and Numbering Statements and Functions](#)

[FormatBullet](#)

[FormatBulletsAndNumbering](#)

FormatHeadingNumber
FormatNumber

FormatNumber

FormatNumber [.Points = number] [, .Color = number] [, .Before = text] [, .Type = number] [, .After = text] [, .StartAt = number] [, .Include = number] [, .Alignment = number] [, .Indent = number or text] [, .Space = number or text] [, .Hang = number] [, .Font = text] [, .Strikethrough = number] [, .Bold = number] [, .Italic = number] [, .Underline = number]

Numbers the selected paragraphs. The arguments for the FormatNumber statement correspond to the options in the Modify Numbered List dialog box (Numbered tab, Bullets And Numbering command, Format menu). You cannot display this dialog box using a Dialog or Dialog() instruction.

Argument	Explanation
.Points, .Color, .Font, .Strikethrough, .Bold, .Italic, .Underline	Apply character formatting to numbers at the specified level. For argument descriptions, see FormatFont .
.Before	The text, if any, you want to appear before each number.
.Type	Specifies a format for numbering lists: 0 (zero) 1, 2, 3, 4 1 I, II, III, IV 2 i, ii, iii, iv 3 A, B, C, D 4 a, b, c, d
.After	The text, if any, you want to appear after each number.
.StartAt	The number for the first selected paragraph. If .Type is 3 or 4, .StartAt corresponds to the position in the alphabet of the starting letter.
.Include	Specifies whether to include numbers and position options from the previous headings for numbers at the specified level: 0 (zero) Includes neither numbers nor position options. 1 Includes a series of numbers from higher-level headings before the numbers at the specified level. 2 Includes both numbers from higher-level headings and position options from the previous level.
.Alignment	Specifies an alignment for the numbers within the space between the left indent and the first line of text; takes effect only if .Space is 0 (zero): 0 (zero) or omitted Left 1 Centered 2 Right
.Indent	The distance between the left indent and the first line of text, in points or a text measurement.
.Space	The distance between the number and the first line of text, in points or a text measurement.
.Hang	If 1, applies a hanging indent to the selected paragraphs.
.Font	The font to apply to the numbers.

See also

[Bullets and Numbering Statements and Functions](#)

FormatBullet

FormatBulletsAndNumbering

FormatHeadingNumber

FormatMultilevel

FormShading, FormShading()

FormShading [On]

FormShading()

The FormShading statement controls shading for form fields in the active document.

Argument	Explanation
On	Specifies whether to display form fields with or without shading.
1	Displays form fields with shading.
0 (zero)	Displays form fields without shading.
Omitted	Toggles form-field shading.

The FormShading() function returns 0 (zero) if form fields are not shaded and -1 if they are.

See also

Forms Statements and Functions

FormFieldOptions

GetDocumentVarName\$()

GetDocumentVarName\$(VariableNumber)

Returns the name of a document variable set with **SetDocumentVar** or **SetDocumentVar()**.

Argument

Explanation

VariableNumber The number of the document variable, from 1 to the total number of document variables stored in the active document (you can obtain the total using **CountDocumentVars()**).

For an example, see [CountDocumentVars\(\) Example](#).

See also

[Documents, Templates, and AddIns Statements and Functions](#)

[CountDocumentVars\(\)](#)

[GetDocumentVar\\$\(\)](#)

[SetDocumentVar](#)

InsertSectionBreak

Inserts a section break with the same formatting as the section containing the insertion point.

See also

Section and Document Formatting Statements and Functions

InsertBreak

InsertColumnBreak

InsertPageBreak

LockDocument, LockDocument()

LockDocument [Lock]

LockDocument()

The LockDocument statement adds or removes read-only protection for an entire master document or one of its subdocuments. If the insertion point is within a master document but not within a subdocument, LockDocument locks or unlocks the entire document. If the insertion point is within a subdocument, LockDocument locks or unlocks the subdocument only.

Argument	Explanation
Lock	Specifies whether to add or remove read-only protection for the subdocument or master document: 0 (zero) Removes read-only protection. Note that if you unlock an entire master document, Word unlocks all subdocuments that were previously locked. 1 Adds read-only protection. Omitted Toggles read-only protection.

The LockDocument() function returns -1 if the subdocument or master document is read-only and 0 (zero) if it is not. Note that when the insertion point is in a subdocument, LockDocument() returns information about the read-only state of the subdocument only, not of the entire master document.

See also

Environment Statements and Functions

ToolsProtectDocument

ToolsProtectSection

ToolsUnprotectDocument

Magnifier, Magnifier()

Magnifier [On]

Magnifier()

The Magnifier statement changes the mouse pointer from the standard pointer to a pointer resembling a magnifying glass, or vice versa, in print preview. When the mouse pointer is a magnifying glass, the user can zoom in on a particular area of the page or zoom out to see an entire page or pages.

Argument	Explanation
On	Specifies the mouse pointer to display in print preview:
0 (zero)	Displays the standard pointer.
1	Displays the magnifying glass pointer.
Omitted	Toggles the mouse pointer.

The Magnifier() function returns -1 if the mouse pointer is a magnifying glass and 0 (zero) if it is the standard pointer.

See also

[View Statements and Functions](#)

[FilePrintPreview](#)

[ViewZoom](#)

MicrosoftSystemInfo

Runs Microsoft System Info, which displays information about the current operating environment.

See also

Environment Statements and Functions

AppInfo\$()

GetSystemInfo

NormalViewHeaderArea

NormalViewHeaderArea [.Type = number] [, .FirstPage = number] [, .OddAndEvenPages = number] [, .HeaderDistance = text] [, .FooterDistance = text]

Opens the header/footer pane (normal and outline views) or displays the header or footer area (page layout view) and sets options for headers and footers. Word version 6.0 preserves the ability to display the header/footer pane so you can edit any type of header or footer, regardless of the number of pages in a document, and so the spelling checker can highlight misspelled words in a header or footer.

The arguments for the NormalViewHeaderArea statement correspond to the options in the Header/Footer dialog box in Word version 2.x. Note that these options are usually set using FilePageSetup in Word version 6.0. Although you can retrieve information from the NormalViewHeaderArea dialog record, you cannot use this statement to display the Word version 2.x dialog box.

Argument	Explanation
.Type	<p>Specifies whether to display the header or footer area. The possible values of .Type depend on the settings of .FirstPage and .OddAndEvenPages.</p> <p>If both .FirstPage and .OddAndEvenPages are set to 0 (zero):</p> <ul style="list-style-type: none">0 (zero) Header1 Footer <p>If .FirstPage is set to 1 and .OddAndEvenPages is set to 0 (zero):</p> <ul style="list-style-type: none">0 (zero) Header1 Footer2 First header3 First footer <p>If .FirstPage is set to 0 (zero) and .OddAndEvenPages is set to 1:</p> <ul style="list-style-type: none">0 (zero) Even header1 Even footer2 Odd header3 Odd footer <p>If both .FirstPage and .OddAndEvenPages are set to 1:</p> <ul style="list-style-type: none">0 (zero) First header1 First footer2 Even header3 Even footer4 Odd header5 Odd footer
.FirstPage	<p>If 1, allows a header or footer for the first page that differs from the rest of the pages in the section.</p>
.OddAndEvenPages	<p>If 1, allows one header or footer for even-numbered pages and a different header or footer for odd-numbered pages.</p>
.HeaderDistance	<p>The distance from the top of the page to the header.</p>
.FooterDistance	<p>The distance from the bottom of the page to the footer.</p>

See also

[View Statements and Functions](#)

[FilePageSetup](#)

[ViewFooter](#)

[ViewHeader](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCclosewc shareres, T3EWClass, \$\$button:WordBASICCcopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

PathFromMacPath\$() Example

In Word for Windows, this example returns the path and filename "\\ HD80\ Reports\ !FinalRe.por".

```
winpath$ = PathFromMacPath$ ("HD80:Reports:Final Report")
```

PathFromMacPath\$()

Example

PathFromMacPath\$(Path\$)

Converts the Macintosh path and filename specified by Path\$ to a valid path and filename for the current operating system.

In Windows, each directory name and filename may contain up to eight characters followed by an optional filename extension (a period and up to three characters). When converting a Macintosh path to a valid Windows path, Word does the following to each Macintosh directory name and filename:

- Removes spaces.
- Adds an exclamation point (!) before the directory name or filename if spaces or extra characters are removed.
- If the directory name or filename is longer than eight characters, adds a period and removes extra characters to form a valid Windows directory name or filename with an extension; for example, the Macintosh directory name "Employee Addresses" becomes the Windows directory name "!Employee.ead".
- Uses the first period, if any, to determine where the extension begins in the Windows directory name or filename, removing any unusable characters; for example, the Macintosh filename "PC text file. text" becomes the Windows filename "!PCtextf.tex".
- If there is more than one period, removes all characters between the first and the last period; for example, the Macintosh filename "chapter1.rev.3" becomes the Windows filename "!chapter.3".

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

SelectionFileName\$() Example

This example checks to see if the active window is a macro-editing window. If not, the example checks the last character in the text returned by SelectionFileName\$(). If the last character is a backslash (\), indicating the document has never been saved, a message is displayed.

```
a$ = SelectionFileName$()
If SelInfo(27) = -1 Then
    MsgBox "A macro-editing window is active."
    Goto bye
End If
If Right$(a$, 1) = "\" Then
    MsgBox "The active document has never been saved."
End If
bye:
```

SelectionFileName\$()

SelectionFileName\$()

~~Example~~ Returns the full path and filename of the active document if it has been saved. If the document has not been saved, or if the active window is a macro-editing window, SelectionFileName\$() returns the current path followed by a backslash (\).

See also

Documents, Templates, and AddIns Statements and Functions

FileName\$()

FileNameInfo\$()

GetDirectory\$()

WW2CallingConvention, WW2CallingConvention()

WW2CallingConvention [On]

WW2CallingConvention()

The WW2CallingConvention statement (Word 6.0a only) controls how Word resolves naming conflicts when one macro calls another. A conflict arises if a macro with the specified name exists both in the active template and the template containing the calling macro. In Word 2.x, the macro in the active template runs. In Word 6.0, the macro in the calling template runs.

By including a WW2CallingConvention instruction at the beginning of a macro, you can temporarily revert to Word 2.x behavior. When the macro containing the WW2CallingConvention instruction ends, Word 6.0 behavior is restored.

Argument	Explanation
On	Specifies how Word resolves naming conflicts: 0 (zero) Word 6.0 behavior (in favor of the calling template) 1 or omitted Word 2.x behavior (in favor of the active template)

The WW2CallingConvention() function returns -1 if the Word 2.x calling convention is in effect and 0 (zero) if it isn't.

In general, you should use WW2CallingConvention only if you already have a suite of templates that rely on Word 2.x behavior to resolve naming conflicts. This statement can be a handy way to get your solution up and running in Word 6.0a without a major rewrite. However, be aware that whoever runs your solution will also need Word 6.0a because WW2CallingConvention is not part of Word 6.0.

See also

[Converting Word Version 2.x Macros](#)

[ToolsMacro](#)

The Microsoft Word Developer's Kit

The Microsoft Word Developer's Kit, published by Microsoft Press, is a comprehensive guide and reference to programming macros in WordBasic. The book is divided into three parts:

- Part 1, "Learning WordBasic," gets you started programming in WordBasic or learning the details of WordBasic if you already know another Basic programming language.
- Part 2, "WordBasic Reference," is a printed version of the statements and functions reference in WordBasic Help.
- Part 3, "Appendixes," provides information about the tools and extensions to WordBasic included on a companion disk.

The disk provided in the Microsoft Word Developer's Kit includes the following:

- Workgroup extensions for WordBasic, which allow access to the messaging application programming interface (MAPI). With Workgroup extensions, you can include electronic mail (e-mail) in your custom applications.
- Open database connectivity (ODBC) extensions for WordBasic, which allow access to data in any database management system (DBMS) that supports the ODBC application programming interface (API) standard.
- Tools for creating add-ins that interact directly with Microsoft Word using the Microsoft Word application programming interface (Word API).
- Templates containing example macros and tools, including a wizard that helps you set up your own custom wizards.

Microsoft Word Developer's Kit (Microsoft Press, 1993) ISBN 1-55615-630-8. Available wherever computer books are sold and directly from Microsoft Press. Credit card orders: 1-800-MS-PRESS or 615-793-5090. CompuServe: GO MSP.

Microsoft Word Developer's Kit (Microsoft Press, 1993) ISBN 1-55615-630-8. Available wherever computer books are sold and directly from Microsoft Press. For more information, including a description of the contents of the Developer's Kit and how to place orders, see [The Microsoft Word Developer's Kit](#).

Microsoft Solution Providers

Microsoft Solution Providers are independent organizations that provide consulting, integration, customization, development, technical support and training, or other services with Microsoft products. These companies are called Solution Providers because they apply technology and provide high-quality services to help solve real-world business problems.

If your organization develops custom solutions using Microsoft Word or Microsoft Office, or if you design, integrate, train, support, or provide other services for Microsoft products, the Microsoft Solution Provider program may be for you. Solution Providers receive business development assistance, access to information and technology, and membership in a powerful community.

To find out more about Microsoft Solution Providers

- In the U.S., call 1-800-426-9400.
- In Canada, call (800) 563-9048.
- Outside North America, contact your local Microsoft office.

Key Examples in WordBasic Help

How to get a list of the files in a directory

You use the Files\$() function to return the list of files in a directory. The trick is first to use Files\$() to specify the list of files to return, and then to use Files\$() within a loop to return the rest of the files in the directory.

Examples

How to display a custom dialog box

You use either the Dialog statement or the Dialog() function to display a custom dialog box. Generally, the Dialog() function is preferred, since it returns the value of the command button chosen. If you use the Dialog statement instead, an error is generated if the user chooses the Cancel button in the dialog box (the error can be trapped using an On Error instruction). Before a custom dialog box can be displayed, a Begin Dialog...End Dialog statement must be used to create a dialog box definition and a Dim statement must be used to create a dialog record.

Examples

How to display a Word dialog box

You use either the Dialog statement or the Dialog() function to display a Word dialog box. If you use the Dialog statement, an error is generated if the user chooses the Cancel button in the dialog box (the error can be trapped using an On Error instruction). Before a Word dialog box can be displayed, a Dim statement must be used to create a dialog record and the GetCurValues statement must be used to place the current values of the dialog box into the dialog record.

Examples

How to retrieve values from a Word dialog box

You can retrieve the value of one or more Word dialog box settings by using the Dim statement to define a dialog record for that dialog box and using the GetCurValues statement to place the current values of the dialog box into the dialog record. You can then use the syntax DialogRecord.ArgumentName to retrieve dialog box values, where ArgumentName is the name of an argument for the WordBasic statement that corresponds to the dialog box.

Example

How to get a list of AutoText entries, bookmarks, or available fonts

Many WordBasic functions beginning with "Count" return the numbers of different items stored in the active document or template. For example, the CountAutoTextEntries() function returns the number of AutoText entries in a template; the CountBookmarks() function returns the number of bookmarks in the active document; and the CountFonts() function returns the number of fonts available on the active printer. You can combine these functions with other functions, such as AutoTextName\$(), BookmarkName\$(), and Font\$() to return lists of AutoText entries, bookmarks, or fonts.

AutoText Entries Example

Bookmarks Examples

Fonts Example

How to work on part of a document

You can use bookmarks and the CmpBookmarks() function to restrict the operation of a macro to a particular part of a document.

Example

How to switch between windows

It is often useful for a macro to switch between active windows. You can use the Activate or WindowList statements to activate a document window or macro-editing window.

Example

How to create "permanent" variables

You can use the SetPrivateProfileString and SetDocumentVar statements to create variables that persist after a macro has finished running. The SetDocumentVar statement creates a document variable in the active document. The SetPrivateProfileString statement creates a variable setting in a settings file stored in the Windows directory.

Document Variable Example

Settings File Example

How to insert text into a document

You use the Insert statement to insert text into a document. The Insert statement can insert into a document anything a user can insert using the keyboard, including nonprinting characters such as tab characters.

Examples

How to retrieve text from a document

Generally, you use the Selection\$() function to return text from a document to a macro. The Selection\$() function returns the text of the current selection. You can also use the GetBookmark\$() function to return bookmarked text in the active document.

Selection\$() Example

GetBookmark\$() Example

Operators and Predefined Bookmarks

Operators

Overview

Operator Precedence

Arithmetic Operators

The String Concatenation Operator

Comparison Operators

Logical Operators

Predefined Bookmarks

Predefined Bookmarks

Overview of Operators

An expression is any valid combination of operators, variables, numbers, strings, and WordBasic functions that can be evaluated to a single result. Depending on the kind of operator and values used, the result of an expression can be a number, string, or logical value, where the numbers -1 and 0 (zero) represent the logical values true and false, respectively. In WordBasic, there are four categories of operators to use with values to form expressions: arithmetic, string concatenation, comparison, and logical. This section describes the operators within these categories in order of operator precedence.

Operator Precedence

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order known as operator precedence. Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before others. Operations within parentheses are always performed before those outside parentheses.

Within parentheses, however, normal operator precedence is maintained. When expressions contain operators from more than one category, arithmetic operators (including the string concatenation operator) are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last.

Within an expression, multiplication and division operations are evaluated before addition and subtraction operations. When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right. All comparison operators have equal precedence; that is, they are evaluated in the left-to-right order in which they appear.

The string concatenation operator (+) is not really an arithmetic operator, but in precedence it does fall after all arithmetic operators and before all comparison operators.

Arithmetic Operators

Use these operators to generate any numeric value to assign to a variable or to use in input, output, or loops.

Operator	Description
- (Negation)	Indicates that the operand is a negative value. The operand can be any numeric expression.
* (Multiplication)	Multiplies two numbers. The operands can be any numeric expressions.
/ (Division)	Divides two numbers. The operands can be any numeric expressions.
MOD (Modular division)	Divides two operands and returns only the remainder. For example, the result of the expression $19 \text{ MOD } 7$ (which can be read as 19 modulo 7) is 5. The operands can be any numeric expressions.
+ (Addition)	Sums two numbers. The operands can be any numeric expressions. Note that you also use + as the string concatenation operator.
- (Subtraction)	Finds the difference between two numbers. The operands can be any numeric expressions.

The String Concatenation Operator

Use the string concatenation operator to link literal strings and string variables.

Operator	Description
+ (String concatenation)	<p>Concatenates two strings. For example, the result of "Microsoft " + "Word" is "Microsoft Word".</p> <p>You must ensure that spaces are included in the strings being concatenated to avoid running words or characters together.</p> <p>If you use the Str\$() function to return numbers as strings, note that the function adds a space before positive numbers (for example, Str\$(47) returns " 47"), but not before negative numbers (for example, Str\$(-47) returns "-47").</p> <p>Note that you also use + as the addition operator.</p>

Comparison Operators

Use these operators, also known as relational operators, to compare two expressions (numeric or string) and return true (-1) or false (0) values for use in control structures such as If conditionals and While... Wend loops. The following table lists the comparison operators and the conditions that determine whether the result is true or false.

Operator	True	False
= (Equal to)	exp1 = exp2	exp1 <> exp2
<> (Not equal to)	exp1 <> exp2	exp1 = exp2
< (Less than)	exp1 < exp2	exp1 >= exp2
> (Greater than)	exp1 > exp2	exp1 <= exp2
<= (Less than or equal to)	exp1 <= exp2	exp1 > exp2
>= (Greater than or equal to)	exp1 >= exp2	exp1 < exp2

Logical Operators

Use these operators in combination with comparison expressions to create compound logical expressions that return true (-1) or false (0) values.

Operator	Description								
AND	<p>If, and only if, both expressions evaluate true, the result is true. If either expression evaluates false, the result is false. The result is determined as follows:</p> <table><tbody><tr><td>True AND True</td><td>True</td></tr><tr><td>False AND True</td><td>False</td></tr><tr><td>True AND False</td><td>False</td></tr><tr><td>False AND False</td><td>False</td></tr></tbody></table>	True AND True	True	False AND True	False	True AND False	False	False AND False	False
True AND True	True								
False AND True	False								
True AND False	False								
False AND False	False								
OR	<p>If either or both expressions evaluate true, the result is true. The result is determined as follows:</p> <table><tbody><tr><td>True OR True</td><td>True</td></tr><tr><td>False OR True</td><td>True</td></tr><tr><td>True OR False</td><td>True</td></tr><tr><td>False OR False</td><td>False</td></tr></tbody></table>	True OR True	True	False OR True	True	True OR False	True	False OR False	False
True OR True	True								
False OR True	True								
True OR False	True								
False OR False	False								
NOT	<p>The result is determined as follows:</p> <table><tbody><tr><td>NOT False</td><td>True</td></tr><tr><td>NOT True</td><td>False</td></tr></tbody></table> <p>Note that a NOT compound expression evaluates as described only when the operands are comparisons or numeric true and false values, where true is -1 and false is 0 (zero).</p>	NOT False	True	NOT True	False				
NOT False	True								
NOT True	False								

Predefined Bookmarks

Example

Word sets and automatically updates a number of reserved bookmarks. You can use these predefined bookmarks just as you use the ones that you place in documents, except that you don't have to set them and they are not listed in the Go To dialog box (Edit menu). The following table describes the predefined bookmarks available in Word.

Bookmark	Description
\Sel	Current selection or the insertion point.
\PrevSel1	Most recent selection where editing occurred; going to this bookmark is equivalent to running the GoBack statement once.
\PrevSel2	Second most recent selection where editing occurred; going to this bookmark is equivalent to running the GoBack statement twice.
\StartOfSel	Start of the current selection.
\EndOfSel	End of the current selection.
\Line	Current line or the first line of the current selection. If the insertion point is at the end of a line that is not the last line in the paragraph, the bookmark includes the entire next line.
\Char	Current character, which is the character following the insertion point if there is no selection, or the first character of the selection.
\Para	Current paragraph, which is the paragraph containing the insertion point or, if more than one paragraph is selected, the first paragraph of the selection. Note that if the insertion point or selection is in the last paragraph of the document, the "\Para" bookmark does not include the paragraph mark.
\Section	Current section, including the break at the end of the section, if any. The current section contains the insertion point or selection. If the selection contains more than one section, the "\Section" bookmark is the first section in the selection.
\Doc	Entire contents of the active document, with the exception of the final paragraph mark.
\Page	Current page, including the break at the end of the page, if any. The current page contains the insertion point. If the current selection contains more than one page, the "\Page" bookmark is the first page of the selection. Note that if the insertion point or selection is in the last page of the document, the "\Page" bookmark does not include the final paragraph mark.

<code>\StartOfDoc</code>	Beginning of the document.
<code>\EndOfDoc</code>	End of the document.
<code>\Cell</code>	Current cell in a table, which is the cell containing the insertion point. If one or more cells of a table are included in the current selection, the " <code>\Cell</code> " bookmark is the first cell in the selection.
<code>\Table</code>	Current table, which is the table containing the insertion point or selection. If the selection includes more than one table, the " <code>\Table</code> " bookmark is the entire first table of the selection, even if the entire table is not selected.
<code>\HeadingLevel</code>	The heading that contains the insertion point or selection, plus any subordinate headings and text. If the current selection is body text, the " <code>\HeadingLevel</code> " bookmark includes the preceding heading, plus any headings and text subordinate to that heading.

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

Predefined Bookmarks Example

The following macro demonstrates a typical use of predefined bookmarks. The macro moves line by line through a document from the current line and removes any leading spaces from the lines. The While...

Wend instruction uses the "\Sel" (current selection) and "\EndOfDoc" bookmarks with the CmpBookmarks() function to determine whether the selection is at the end of the document. When the end of the document is reached, Word displays a message to alert the user.

```
Sub MAIN
StartOfLine
While CmpBookmarks("\Sel", "\EndOfDoc")
  A$ = GetBookmark$("\Line")
  B = Asc(A$)
  If B = 32 Then DeleteWord
  EndOfLine
  CharRight
Wend
MsgBox "End of document."
End Sub
```

The CmpBookmarks() function compares two bookmarks and can return a number of different values according to the relative location and size of the bookmarks.

For other examples of predefined bookmarks used in WordBasic macros, see CmpBookmarks(), CopyBookmark, ParaDown, Select Case.

Conventions

In the Help topic for each WordBasic statement or function, the statement or function name appears as a bold heading at the top of the window. One or more syntax statements follow the bold heading. Here is a syntax example:

CharLeft [Count] [, Select]

When you type an instruction, you must include all the items in the syntax that are formatted in bold. Items enclosed in brackets are optional. Do not type the brackets when including an optional item. Italic formatting indicates argument names or value placeholders that you replace with actual values or variables to which you've already assigned values.

For example, you could use any of the following CharLeft instructions in a macro:

```
CharLeft
CharLeft 1
CharLeft 1, 1
```

If you assigned acceptable values to the numeric variables `move` and `extend`, you could use the following CharLeft instruction:

```
CharLeft move, extend
```

Note that you must separate arguments with commas. The acceptable values for arguments are listed in the remarks following the syntax, usually in a table. Some syntax examples include required arguments. For example:

EditReplaceStyle .Style = text

To use this statement, you must include the .Style argument---note the period preceding the argument name. You must type all the text that appears in bold and supply a specific value or variable for the italic placeholder, as in the following examples:

```
EditReplaceStyle .Style = "Heading 1"
EditReplaceStyle .Style = "Normal"
```

Other statements and functions include a mixture of required and optional arguments:

EditAutoText .Name = text [, .Context = number] [, .InsertAs = number] [, .Insert] [, .Add] [, .Delete]

According to this syntax, you must include the first argument and a value, but the remaining arguments are optional. As the syntax indicates, every argument in your final macro instruction must be separated by a comma. For example:

```
EditAutoText .Name = "disclaimer", .Context = 1, .Add
```

Most topics in WordBasic Help include examples of how to use specific statements and functions.

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

AppActivate Example

This example activates File Manager if it is running and starts File Manager if it is not running:

```
If AppIsRunning("File Manager") Then
    AppActivate "File Manager"
Else
    Shell "WINFILE.EXE"
End If
```

AppActivate

Example

AppActivate WindowName\$ [, Immediate]

Activates a running application.

Argument	Explanation
WindowName\$	The name of the application window to activate, as it appears in the title bar or Task List. It is not necessary to specify the entire window name. For example, to indicate a window named "Notepad - FILES.TXT," you can specify "Notepad - FILES.TXT," "Notepad," or even "Note." The first window name in the Task List that matches the beginning of the specified string is affected. The case of characters is not significant in WindowName\$.
Immediate	Specifies when to switch to the other application: 0 (zero) or omitted If Word is not active, Word flashes its title bar or icon, waits for the user to activate Word, and then activates the other application. 1 Word immediately activates the other application, even if Word is not the active application.

See also

Application Control Statements and Functions

AppClose

AppGetNames

AppIsRunning()

MicrosoftAccess

MicrosoftExcel

MicrosoftFoxPro

MicrosoftMail

MicrosoftPowerPoint

MicrosoftProject

MicrosoftPublisher

MicrosoftSchedule

Shell

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

AppClose Example

This example closes Microsoft Excel if it is running:

```
If AppIsRunning("Microsoft Excel") Then
    AppClose "Microsoft Excel"
End If
```

AppClose

Example

AppClose [WindowName\$]

Closes the specified application.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see <u>AppActivate</u> .

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

Shell

AppCount()

AppCount()

Returns the number of open applications (including hidden applications that do not appear in the Task List). For an example, see [AppGetNames Example](#).

See also

[Application Control Statements and Functions](#)

[AppGetNames](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

AppGetNames Example

This example inserts a list of application window names at the insertion point:

```
size = AppCount() - 1
Dim winnames$(size)
AppGetNames winnames$()
For i = 0 To size
    Insert winnames$(i)
    InsertPara
Next
```

AppGetNames, AppGetNames()

Example

AppGetNames ArrayVariable\$()

AppGetNames(ArrayVariable\$())

The AppGetNames statement fills a previously defined string array with the names of open application windows (including hidden application windows that do not appear in the Task List). If ArrayVariable\$() has fewer elements than the number of open applications, the array is filled with as many names as there are elements, and an error does not occur.

The AppGetNames() function carries out the same action and also returns the number of open application windows (including hidden application windows that do not appear in the Task List). AppGetNames() returns the same value as AppCount().

See also

Application Control Statements and Functions

AppActivate

AppClose

AppCount()

AppIsRunning

AppHide

AppHide [WindowName\$]

Hides the specified application and removes its window name from the Task List.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see AppActivate .

See also

[Application Control Statements and Functions](#)

[AppClose](#)

[AppShow](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

AppInfo\$() Example

This example displays a message box containing the version number of Word:

```
ver$ = AppInfo$(2)  
MsgBox ver$, "Microsoft Word Version", 64
```

AppInfo\$()

Example

AppInfo\$(Type)

Returns one of 25 types of information about the Word application. Note that the GetSystemInfo\$() function returns similar information. Also, you can use the GetSystemInfo statement to fill an array with system information.

Type is one of the following numeric codes, specifying the type of information to return.

Type	Explanation
1	Environment (for example, "Windows 3.10").
2	Word version number (for example, "6.0").
3	Returns -1 if Word is in a special mode (for example, CopyText or MoveText mode).
4	Distance from the left edge of the screen to the left border of the Word window, in points (72 points = 1 inch). Note that when Word is maximized, AppInfo\$(4) returns a negative value to indicate the borders are beyond the edge of the screen (this value varies depending on the width of the borders).
5	Distance from the top of the screen to the top border of the Word window, in points. Note that when Word is maximized, AppInfo\$(5) returns a negative value to indicate the borders are beyond the edge of the screen (this value varies depending on the width of the borders).
6	Width of the workspace, in points; the width increases as you hide Word screen elements or widen the Word window. Note that increasing the zoom percentage decreases the return value and vice versa.
7	Height of the workspace, in points; the height increases as you hide Word screen elements or increase the height of the Word window. Note that increasing the zoom percentage decreases the return value and vice versa.
8	Returns -1 if the application is maximized.
9	Total conventional memory, in kilobytes.
10	Available conventional memory, in kilobytes.
11	Total expanded memory, in kilobytes.
12	Available expanded memory, in kilobytes.
13	Returns -1 if a math coprocessor is installed.
14	Returns -1 if a mouse is installed.

- 15 Available disk space, in kilobytes.
- 16 Returns the language version of Word. For example, returns "Français" for the French version of Word. For a list of languages, see [ToolsLanguage](#).
- 17 Returns the list separator setting ("sList") in the [intl] section of WIN.INI.
- 18 Returns the decimal setting ("sDecimal") in the [intl] section of WIN.INI.
- 19 Returns the thousand separator ("sThousand") in the [intl] section of WIN.INI.
- 20 Returns the currency symbol ("sCurrency") in the [intl] section of WIN.INI.
- 21 Returns the clock format ("iTime") in the [intl] section of WIN.INI.
- 22 Returns the A.M. string ("s1159") in the [intl] section of WIN.INI.
- 23 Returns the P.M. string ("s2359") in the [intl] section of WIN.INI.
- 24 Returns the time separator ("sTime") in the [intl] section of WIN.INI.
- 25 Returns the date separator ("sDate") in the [intl] section of WIN.INI.

See also

[Application Control Statements and Functions](#)

[AppGetNames](#)

[GetSystemInfo](#)

AppIsRunning()

AppIsRunning(WindowName\$)

Returns -1 if the specified application is running or 0 (zero) if it is not.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. For more information on WindowName\$, see AppActivate .

For an example, see [AppActivate Example](#).

See also

[Application Control Statements and Functions](#)

[AppActivate](#)

[AppClose](#)

AppMaximize, AppMaximize()

AppMaximize [WindowName\$] [, State]

AppMaximize([WindowName\$])

The AppMaximize statement maximizes or restores the specified application.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see AppActivate .
State	Specifies whether to maximize or restore the application: 0 (zero) Restores the application. 1 Maximizes the application. Omitted Toggles between restored and maximized states. If the state of the application changes, it is activated. If the state does not change (for example, if you run the instruction <code>AppMaximize "Microsoft Excel", 1</code> and Microsoft Excel is already maximized), the application is not activated.

The AppMaximize() function returns the following values.

Value	Explanation
-1	If the application is maximized
0 (zero)	If the application is not maximized

See also

[Application Control Statements and Functions](#)

[AppMinimize](#)

[AppMove](#)

[AppRestore](#)

[AppSize](#)

[DocMaximize](#)

AppMinimize, AppMinimize()

AppMinimize [WindowName\$] [, State]

AppMinimize([WindowName\$])

The AppMinimize statement minimizes or restores the specified application.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see AppActivate .
State	Specifies whether to minimize or restore the application: 0 (zero) Restores the application. 1 Minimizes the application. Omitted Toggles between restored and minimized states. If the application is restored from an icon, it is activated. If the state does not change or if the application is minimized, the application is not activated.

Note

If an untrapped error occurs in a macro while Word is minimized, the macro halts and the Word icon flashes. When Word is restored, it displays a message indicating the nature of the error.

The AppMinimize() function returns the following values.

Value	Explanation
-1	If the application is minimized
0 (zero)	If the application is not minimized

See also

[Application Control Statements and Functions](#)

[AppMaximize](#)

[AppMove](#)

[AppRestore](#)

[AppSize](#)

[DocMinimize](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

AppMove Example

This example starts Microsoft Excel if it is not running and then arranges Word and Microsoft Excel into nonoverlapping windows, each one-half the height of the screen:

```
If AppIsRunning("Microsoft Excel") = 0 Then MicrosoftExcel
AppRestore
AppMove 0, 0
AppSize 480, 180
AppRestore "Microsoft Excel"
AppMove "Microsoft Excel", 0, 180
AppSize "Microsoft Excel", 480, 180
```

AppMove

Example

AppMove [WindowName\$,] HorizPos, VertPos

Moves the specified application window or icon to a position relative to the upper-left corner of the screen. If the application is maximized, an error occurs.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window or icon name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see <u>AppActivate</u> .
HorizPos, VertPos	The horizontal (HorizPos) and vertical (VertPos) distance from the upper-left corner of the screen to the upper-left corner of the application window or icon, in points (72 points = 1 inch). Negative measurements are allowed only if you specify WindowName\$.

See also

Application Control Statements and Functions

AppRestore

AppSize

AppWindowPosLeft

AppWindowPosTop

DocMove

AppRestore, AppRestore()

AppRestore [WindowName\$]

AppRestore([WindowName\$])

The AppRestore statement restores the specified application from a maximized or minimized state and activates the application. If the specified application is already restored, AppRestore has no effect.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see AppActivate .

The AppRestore() function returns the following values.

Value	Explanation
-1	If the application is restored
0 (zero)	If the application is not restored

For an example, see [AppMove Example](#).

See also

[Application Control Statements and Functions](#)

[AppMaximize](#)

[AppMinimize](#)

[AppMove](#)

[AppSize](#)

[DocRestore](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

AppSendMessage Example

This example starts the Windows Help application and then sends it a message that displays the Open dialog box. The number 273 is the decimal value associated with the message WM_COMMAND and 1101 is the parameter that specifies the Open command. Lparam is ignored in this case, but must still be specified as 0 (zero).

```
Shell "WINHELP.EXE"
```

```
AppSendMessage "Windows Help", 273, 1101, 0
```

AppSendMessage

Example

AppSendMessage [WindowName\$,] Message, Wparam, Lparam

Sends a Windows message and its associated parameters to the application specified by WindowName\$.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see AppActivate .
Message	A decimal number corresponding to the message you want to send. If you have the Microsoft Windows 3.1 Software Development Kit, you can look up the name of the message in WINDOWS.H and then convert the associated hexadecimal number to a decimal number using Calculator.
Wparam, Lparam	Parameters appropriate for the message you are sending. For information on what these values represent, see the reference topic for the message in the Microsoft Windows 3.1 Programmer's Reference, Volume 3, available in the Microsoft Windows 3.1 Software Development Kit or from Microsoft Press. To retrieve the appropriate values, you may need to use the Spy utility (which comes with the Microsoft Windows 3.1 SDK).

See also

[Application Control Statements and Functions](#)

[AppActivate](#)

[AppIsRunning](#)

[DDEExecute](#)

[DDEPoke](#)

AppShow

AppShow [WindowName\$]

Makes visible and activates an application previously hidden with AppHide and restores the application window name to the Task List. If the application is not hidden, AppShow has no effect.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it would appear in the title bar or Task List if the application were visible. If omitted, Word is assumed. For more information on WindowName\$, see AppActivate .

See also

[Application Control Statements and Functions](#)

[AppActivate](#)

[AppHide](#)

AppSize

AppSize [WindowName\$,] Width, Height

Sizes an application window to a specified width and height. If the application is maximized or minimized, an error occurs.

Argument	Explanation
WindowName\$	A string that matches the beginning of an application window name, as it appears in the title bar or Task List. If omitted, Word is assumed. For more information on WindowName\$, see AppActivate .
Width, Height	The width and height of the application window, in points (72 points = 1 inch).

For an example, see [AppMove Example](#).

See also

[Application Control Statements and Functions](#)

[AppMove](#)

[AppRestore](#)

[AppWindowHeight](#)

[AppWindowWidth](#)

[DocSize](#)

AppWindowHeight, AppWindowHeight()

AppWindowHeight [WindowName\$,] Height

AppWindowHeight([WindowName\$])

The AppWindowHeight statement adjusts the height of an application window to a specified number of points (if WindowName\$ is omitted, Word is assumed). AppWindowHeight allows you to change the height of a window without affecting its width (unlike AppSize). The AppWindowHeight() function returns the height of an application window, in points. For argument descriptions, see AppSize.

See also

Application Control Statements and Functions

AppSize

AppWindowPosLeft

AppWindowPosTop

AppWindowWidth

AppWindowPosLeft, AppWindowPosLeft()

AppWindowPosLeft [WindowName\$,] HorizPos

AppWindowPosLeft([WindowName\$])

The AppWindowPosLeft statement moves an application window or icon to a horizontal position specified in points (if WindowName\$ is omitted, Word is assumed). AppWindowPosLeft allows you to change the horizontal position of a window or icon without affecting its vertical position (unlike AppMove). The AppWindowPosLeft() function returns the horizontal position of an application window or icon, in points. For argument descriptions, see [AppMove](#).

See also

[Application Control Statements and Functions](#)

[AppMove](#)

[AppWindowHeight](#)

[AppWindowPosTop](#)

[AppWindowWidth](#)

AppWindowPosTop, AppWindowPosTop()

AppWindowPosTop [WindowName\$,] VertPos

AppWindowPosTop([WindowName\$])

The AppWindowPosTop statement moves an application window or icon to a vertical position specified in points (if WindowName\$ is omitted, Word is assumed). AppWindowPosTop allows you to change the vertical position of a window or icon without affecting its horizontal position (unlike AppMove). The AppWindowPosTop() function returns the vertical position of an application window or icon, in points. For argument descriptions, see [AppMove](#).

See also

[Application Control Statements and Functions](#)

[AppMove](#)

[AppWindowHeight](#)

[AppWindowPosLeft](#)

[AppWindowWidth](#)

AppWindowWidth, AppWindowWidth()

AppWindowWidth [WindowName\$,] Width

AppWindowWidth([WindowName\$])

The AppWindowWidth statement adjusts the width of an application window to a specified number of points (if WindowName\$ is omitted, Word is assumed). AppWindowWidth allows you to change the width of a window without affecting its height (unlike AppSize). The AppWindowWidth() function returns the width of an application window, in points. For argument descriptions, see [AppSize](#).

See also

[Application Control Statements and Functions](#)

[AppSize](#)

[AppWindowHeight](#)

[AppWindowPosLeft](#)

[AppWindowPosTop](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

ControlRun Example

This example runs the Control Panel:

```
ControlRun .Application = 1
```

ControlRun

Example

ControlRun .Application = number

Runs either the Clipboard or the Control Panel (Windows). If you want to run a different program, use the Shell statement.

Argument	Explanation
.Application	The application to run:
0 (zero)	Clipboard
1	Control Panel

See also

Application Control Statements and Functions

Shell

ExitWindows

ExitWindows

Closes all open applications and quits Windows. ExitWindows does not save changes or prompt you to save changes in Word documents; it does prompt you to save changes in other open applications.

See also

Application Control Statements and Functions

FileExit

FileExit

FileExit [Save]

Quits Word.

Argument	Explanation
Save	Determines whether Word saves each document before closing it if it is "dirty" --- that is, if changes have been made since the last time the file was saved: 0 (zero) or omitted Prompts the user to save each changed document. 1 Saves all edited documents before quitting. 2 Quits without saving changed documents.

See also

Application Control Statements and Functions

AppClose

ExitWindows

FileCloseAll

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

GetSystemInfo Examples

This example creates a table of system information in a new document. First, the example defines and fills an array with labels for each type of system information. Second, the example opens a new document and defines the `info$()` array, which `GetSystemInfo` then fills with the system information. Finally, the `For...` Next loop inserts the table of information.

```
Dim a$(11)
a$(0) = "Environment" : a$(1) = "CPU" : a$(2) = "MS-DOS"
a$(3) = "Windows" : a$(4) = "% Resources" : a$(5) = "Disk Space"
a$(6) = "Mode" : a$(7) = "Coprocessor" : a$(8) = "Country"
a$(9) = "Language" : a$(10) = "Pixels High" : a$(11) = "Pixels Wide"
Dim info$(11)
GetSystemInfo info$()
FileNewDefault
FormatTabs .Position = "1.5 in", .Set
For i = 0 To 11
    Insert a$(i) + Chr$(9) + info$(i)
    InsertPara
Next
```

The following example displays in a message box the amount of available disk space:

```
space$ = GetSystemInfo$(26)
MsgBox "Available disk space: " + space$ + " bytes."
```

GetSystemInfo, GetSystemInfo\$()

Example

GetSystemInfo Array\$()

GetSystemInfo\$(Type)

The GetSystemInfo statement fills a previously defined string array with information about the environment in which Word is running.

The GetSystemInfo\$() function returns one piece of information about the environment in which Word is running. Type is one of the following numeric codes, specifying the type of information to return.

Type	Explanation
21	The environment (for example, "Windows" or "Windows NT")
22	The type of central processing unit, or CPU (for example, "80286," "80386," "i486," or "Unknown")
23	The MS-DOS version number
24	The Windows version number
25	The percent of system resources available
26	The amount of available disk space, in bytes
27	The mode under which Windows is running: "Standard" or "386-Enhanced"
28	Whether a math coprocessor is installed: "Yes" or "No"
29	The country setting ("iCountry") in the [intl] section of WIN.INI
30	The language setting ("sLanguage") in the [intl] section of WIN.INI
31	The vertical display resolution, in pixels
32	The horizontal display resolution, in pixels

See also

Application Control Statements and Functions

AppInfo\$()

MicrosoftAccess

Starts Microsoft Access if it is not running or switches to Microsoft Access if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftExcel

MicrosoftFoxPro

MicrosoftMail

MicrosoftPowerPoint

MicrosoftProject

MicrosoftPublisher

MicrosoftSchedule

MicrosoftExcel

Starts Microsoft Excel if it is not running or switches to Microsoft Excel if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftAccess

MicrosoftFoxPro

MicrosoftMail

MicrosoftPowerPoint

MicrosoftProject

MicrosoftPublisher

MicrosoftSchedule

MicrosoftFoxPro

Starts Microsoft FoxPro if it is not running or switches to Microsoft FoxPro if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftAccess

MicrosoftExcel

MicrosoftMail

MicrosoftPowerPoint

MicrosoftProject

MicrosoftPublisher

MicrosoftSchedule

MicrosoftMail

Starts Microsoft Mail if it is not running or switches to Microsoft Mail if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftAccess

MicrosoftExcel

MicrosoftFoxPro

MicrosoftPowerPoint

MicrosoftProject

MicrosoftPublisher

MicrosoftSchedule

MicrosoftPowerPoint

Starts Microsoft PowerPoint if it is not running or switches to Microsoft PowerPoint if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftAccess

MicrosoftExcel

MicrosoftFoxPro

MicrosoftMail

MicrosoftProject

MicrosoftPublisher

MicrosoftSchedule

MicrosoftProject

Starts Microsoft Project if it is not running or switches to Microsoft Project if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftAccess

MicrosoftExcel

MicrosoftFoxPro

MicrosoftMail

MicrosoftPowerPoint

MicrosoftPublisher

MicrosoftSchedule

MicrosoftPublisher

Starts Microsoft Publisher if it is not running or switches to Microsoft Publisher if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftAccess

MicrosoftExcel

MicrosoftFoxPro

MicrosoftMail

MicrosoftPowerPoint

MicrosoftProject

MicrosoftSchedule

MicrosoftSchedule

Starts Microsoft Schedule+ if it is not running or switches to Microsoft Schedule+ if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

MicrosoftAccess

MicrosoftExcel

MicrosoftFoxPro

MicrosoftMail

MicrosoftPowerPoint

MicrosoftProject

MicrosoftPublisher

RunPrintManager

Starts Print Manager (Windows) if it is not running or switches to Print Manager if it is already running.

See also

Application Control Statements and Functions

AppActivate

AppIsRunning()

ControlRun

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

Shell Examples

This example starts Notepad and loads the document TORT.TXT:

```
Shell "Notepad TORT.TXT"
```

The following example starts Microsoft Excel as a minimized window:

```
Shell "EXCEL.EXE", 2
```

The following example creates a text-only file (DOCLIST.TXT) that lists documents with the filename extension .DOC in the C:\WINWORD directory. You might use an instruction like this to create a file you can open later for sequential input. The "/c" switch ensures that control is returned to Word after the command line following "/c" is run.

```
Shell Environ$("COMSPEC") + "/c dir /b C:\WINWORD\*.DOC > DOCLIST.TXT"
```

Shell

Example

Shell Application\$ [, WindowStyle]

Starts another application (such as Microsoft Excel) or process (such as a batch file or executable file) in Windows.

Argument	Explanation
Application\$	The path and filename required to find the application, as well as any valid switches or arguments you choose to include, just as you would type them in the Run dialog box in Program Manager. Application\$ can be a document filename by itself, provided the filename extension is registered in the [Extensions] section of the WIN.INI file. Shell starts the associated application and opens the document. To display an MS-DOS window, specify <code>Environ\$ ("COMSPEC")</code> as Application\$.
WindowStyle	How the window containing the application should be displayed (some applications ignore this): 0 (zero) Minimized window (icon) 1 Normal window (current window size, or previous size if minimized) 2 Minimized window (for Microsoft Excel compatibility) 3 Maximized window 4 Deactivated window

See also

Application Control Statements and Functions

AppActivate

DDEInitiate()

Environ\$()

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

GetAutoCorrect\$() Example

This example checks the replacement text for the AutoCorrect entry "uk." If the replacement text doesn't match "United Kingdom," the AutoCorrect entry is modified to do so.

```
If GetAutoCorrect$("uk") <> "United Kingdom" Then
    ToolsAutoCorrect .Replace = "uk", \
        .With = "United Kingdom", .Add
End If
```


GetAutoCorrect\$()

Example

GetAutoCorrect\$(AutoCorrectEntry\$)

Returns the replacement text for the specified entry in the Replace column of the AutoCorrect dialog box (Tools menu). If AutoCorrectEntry\$ doesn't exist, GetAutoCorrect\$() returns an empty string (" ").

Argument	Explanation
AutoCorrectEntry\$	The text specified in the Replace column for an AutoCorrect entry in the AutoCorrect dialog box. AutoCorrectEntry\$ is not case-sensitive. For example, you can specify an entry "GW" as either "GW" or "gw."

See also

[AutoCorrect Statements and Functions](#)

[ToolsAutoCorrect](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

ToolsAutoCorrect Example

This example adds a replacement entry and activates automatic replacement of text:

```
ToolsAutoCorrect .ReplaceText = 1, .Replace = "sr", \  
    .With = "Stella Richards", .Add
```

ToolsAutoCorrect

Example

`ToolsAutoCorrect [.SmartQuotes = number] [, .InitialCaps = number] [, .SentenceCaps = number] [, .Days = number] [, .ReplaceText = number] [, .Formatting = number] [, .Replace = text] [, .With = text] [, .Add] [, .Delete]`

Sets AutoCorrect options. The arguments for the ToolsAutoCorrect statement correspond to the options in the AutoCorrect dialog box (Tools menu).

Argument	Explanation
<code>.SmartQuotes</code>	If 1, Word inserts "smart" quotation marks (" " and ' ') and apostrophes (').
<code>.InitialCaps</code>	If 1, Word corrects words in which the first two letters are capitalized. For example, "WORD" becomes "Word."
<code>.SentenceCaps</code>	If 1, Word capitalizes the first letter of new sentences.
<code>.Days</code>	If 1, Word capitalizes the days of the week. For example, "tuesday" becomes "Tuesday."
<code>.ReplaceText</code>	If 1, activates automatic replacement of text.
<code>.Formatting</code>	If 1, formatting is stored with the replacement text when a replacement entry is added; available only if text is selected before running ToolsAutoCorrect.
<code>.Replace</code>	The text you want to replace automatically with the text specified by <code>.With</code> (for example, a person's initials).
<code>.With</code>	The text you want to insert automatically when the text specified by <code>.Replace</code> is typed (for example, a person's full name).
<code>.Add</code>	Adds the text specified by <code>.Replace</code> and <code>.With</code> to the list of replacement entries.
<code>.Delete</code>	Deletes the replacement entry specified by <code>.Replace</code> .

See also

[AutoCorrect Statements and Functions](#)

[ToolsAutoCorrectDays](#)

[ToolsAutoCorrectInitialCaps](#)

[ToolsAutoCorrectReplaceText](#)

[ToolsAutoCorrectSentenceCaps](#)

[ToolsAutoCorrectSmartQuotes](#)

ToolsAutoCorrectDays, ToolsAutoCorrectDays()

ToolsAutoCorrectDays [On]

ToolsAutoCorrectDays()

The ToolsAutoCorrectDays statement selects or clears the Capitalize Names Of Days check box in the AutoCorrect dialog box (Tools menu).

Argument	Explanation
On	Specifies whether to select or clear the check box: 1 Selects the check box. 0 (zero) Clears the check box. Omitted Toggles the check box.

The ToolsAutoCorrectDays() function returns the following values.

Value	Explanation
0 (zero)	If the Capitalize Names Of Days check box is cleared
-1	If the Capitalize Names Of Days check box is selected

See also

[AutoCorrect Statements and Functions](#)

[ToolsAutoCorrect](#)

ToolsAutoCorrectInitialCaps, ToolsAutoCorrectInitialCaps()

ToolsAutoCorrectInitialCaps [On]

ToolsAutoCorrectInitialCaps()

The ToolsAutoCorrectInitialCaps statement selects, clears, or toggles the Correct TWo INitial CApitals check box in the AutoCorrect dialog box (Tools menu). The ToolsAutoCorrectInitialCaps() function returns information about the state of the check box. For information on arguments and return values, see [ToolsAutoCorrectDays](#).

See also

[AutoCorrect Statements and Functions](#)

[ToolsAutoCorrect](#)

[ToolsAutoCorrectDays](#)

`ToolsAutoCorrectReplaceText`, `ToolsAutoCorrectReplaceText()`

`ToolsAutoCorrectReplaceText` [On]

`ToolsAutoCorrectReplaceText()`

The `ToolsAutoCorrectReplaceText` statement selects, clears, or toggles the Replace Text As You Type check box in the AutoCorrect dialog box (Tools menu). The `ToolsAutoCorrectReplaceText()` function returns information about the state of the check box. For information on arguments and return values, see [ToolsAutoCorrectDays](#).

See also

[AutoCorrect Statements and Functions](#)

[ToolsAutoCorrect](#)

[ToolsAutoCorrectDays](#)

ToolsAutoCorrectSentenceCaps, ToolsAutoCorrectSentenceCaps()

ToolsAutoCorrectSentenceCaps [On]

ToolsAutoCorrectSentenceCaps()

The ToolsAutoCorrectSentenceCaps statement selects, clears, or toggles the Capitalize First Letter Of Sentences check box in the AutoCorrect dialog box (Tools menu). The ToolsAutoCorrectSentenceCaps() function returns information about the state of the check box. For information on arguments and return values, see [ToolsAutoCorrectDays](#).

See also

[AutoCorrect Statements and Functions](#)

[ToolsAutoCorrect](#)

[ToolsAutoCorrectDays](#)

ToolsAutoCorrectSmartQuotes, ToolsAutoCorrectSmartQuotes()

ToolsAutoCorrectSmartQuotes [On]

ToolsAutoCorrectSmartQuotes()

The ToolsAutoCorrectSmartQuotes statement selects, clears, or toggles the Change 'Straight Quotes' To 'Smart Quotes' check box in the AutoCorrect dialog box (Tools menu). The ToolsAutoCorrectSmartQuotes() function returns information about the state of the check box. For information on arguments and return values, see [ToolsAutoCorrectDays](#).

See also

[AutoCorrect Statements and Functions](#)

[ToolsAutoCorrect](#)

[ToolsAutoCorrectDays](#)

AutoText

Displays the AutoText dialog box if there is a selection (and proposes up to the first 32 characters of the selection for the unique entry name) or, if there is no selection, attempts to match the text before or surrounding the insertion point with an AutoText entry and insert the entry (including its formatting, if any). Word looks for the entry first in the active template, then in the Normal template, and finally in each loaded global template in the order listed in the Templates And Add-ins dialog box (File menu). If no match can be made, an error occurs. AutoText corresponds to the AutoText button on the Standard toolbar.

See also

AutoText Statements and Functions

AutoTextName\$()

CountAutoTextEntries()

EditAutoText

GetAutoText\$()

InsertAutoText

SetAutoText

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

AutoTextName\$() Example

This example creates a new document that lists all AutoText entries in the Normal template and any loaded global templates. Entry names are inserted with bold formatting and are followed by the contents of the entry.

```
FileNewDefault
For count = 1 To CountAutoTextEntries()
  a$ = AutoTextName$(count)
  Bold 1 : Insert a$
  InsertPara
  Bold 0 : EditAutoText .Name = a$, .Insert
  InsertPara : InsertPara
Next
```

AutoTextName\$()

Example

AutoTextName\$(Count [, Context])

Returns the name of an AutoText entry in the specified context.

Argument	Explanation
Count	The number of the AutoText entry, from 1 to the total number of AutoText entries defined in the given context (you can obtain the total using CountAutoTextEntries()). AutoText entries are listed in alphabetic order.
Context	The context in which to return the name of an AutoText entry: 0 (zero) or omitted Normal template and any loaded global templates 1 Active template Note that if Context is 1 and the active template is the Normal template, AutoTextName\$() generates an error.

See also

AutoText Statements and Functions

AutoText

CountAutoTextEntries()

EditAutoText

GetAutoText\$()

InsertAutoText

SetAutoText

CountAutoTextEntries()

CountAutoTextEntries([Context])

Returns the number of AutoText entries defined for the specified context.

Argument	Explanation
Context	The context in which to count AutoText entries: 0 (zero) or omitted Normal template and any loaded global templates 1 Active template Note that if Context is 1 and the active template is the Normal template, CountAutoTextEntries() returns 0 (zero).

For an example, see [AutoTextName\\$\(\) Example](#).

See also

[AutoText Statements and Functions](#)

[AutoTextName\\$\(\)](#)

[GetAutoText\\$\(\)](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

EditAutoText Examples

This example selects the text of the first paragraph (not including the paragraph mark) and then defines it as an AutoText entry named "MainHead," stored in the Normal template:

```
StartOfDocument
EditGoTo "\Para"
CharLeft 1, 1
EditAutoText .Name = "MainHead", .Context = 0, .Add
```

The following example inserts the "MainHead" AutoText entry without formatting:

```
EditAutoText .Name = "MainHead", .InsertAs = 1, .Insert
```

EditAutoText

Example

EditAutoText .Name = text [, .Context = number] [, .InsertAs = number] [, .Insert] [, .Add] [, .Delete]

Inserts, adds, or deletes an AutoText entry. The arguments for the EditAutoText statement correspond to the options in the AutoText dialog box (Edit menu).

Argument	Explanation
.Name	The name of the AutoText entry.
.Context	A context for the new AutoText entry: 0 (zero) or omitted Normal template 1 Active template Note that .Context is used only when Word adds an AutoText entry. When inserting or deleting an entry, Word automatically looks for the entry first in the active template and then in the Normal template. When inserting an entry and no match is found in the active or Normal templates, Word looks in each loaded global template in the order listed in the Templates And Add-ins dialog box (File menu). You cannot delete an AutoText entry from a loaded global template.
.InsertAs	Used with .Insert to control whether the entry is inserted with its formatting: 0 (zero) or omitted Entry is inserted with formatting. 1 Entry is inserted as plain text.

You can specify only one of the following arguments.

Argument	Explanation
.Insert	Inserts the entry into the document
.Add	Stores the entry in the template (if there is no selection, an error occurs)
.Delete	Deletes the entry from the template

If you do not specify .Insert, .Add, or .Delete, Word inserts the AutoText entry.

See also

[AutoText Statements and Functions](#)

[AutoText](#)

[AutoTextName\\$\(\)](#)

[CountAutoTextEntries\(\)](#)

[GetAutoText\\$\(\)](#)

[InsertAutoText](#)

[SetAutoText](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

GetAutoText\$() Example

This example displays a message box containing the text of the AutoText entry named "Welcome," which is stored in the active template:

```
MsgBox GetAutoText$("Welcome", 1)
```


GetAutoText\$()

Example

GetAutoText\$(Name\$ [, Context])

Returns the unformatted text of the specified AutoText entry.

Argument	Explanation
Name\$	The name of the AutoText entry
Context	Where the AutoText entry is stored: 0 (zero) or omitted Normal template and any loaded global templates 1 Active template

Note that if Context is 1 and the active template is the Normal template, GetAutoText\$() returns an empty string (" ").

See also

AutoText Statements and Functions

AutoText

AutoTextName\$()

CountAutoTextEntries()

EditAutoText

InsertAutoText

SetAutoText

InsertAutoText

InsertAutoText

Attempts to match the current selection or the text before or surrounding the insertion point with an AutoText entry and insert the entry (including its formatting, if any). Word looks for the entry first in the active template, then in the Normal template, and finally in each loaded global template in the order listed in the Templates And Add-ins dialog box (File menu). If no match can be made, an error occurs.

See also

[AutoText Statements and Functions](#)

[AutoText](#)

[AutoTextName\\$\(\)](#)

[CountAutoTextEntries\(\)](#)

[EditAutoText](#)

[GetAutoText\\$\(\)](#)

[SetAutoText](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

SetAutoText Example

This example defines the AutoText entry "Disclaim" in the active template; "Disclaim" contains the text assigned to `text$`:

```
text$ = "No warranty is either expressed or implied."  
SetAutoText "Disclaim", text$, 1
```

SetAutoText

Example

SetAutoText Name\$, Text\$ [, Context]

Defines a text-only AutoText entry. Unlike an EditAutoText instruction that uses .Add, SetAutoText does not require a selection.

Argument	Explanation
Name\$	The name of the new entry.
Text\$	The text to be associated with the entry.
Context	Specifies the availability of the entry: 0 (zero) or omitted Normal template (available to all documents) 1 Active template (available only to documents based on the active template) Note that if Context is 1 and the active template is the Normal template, SetAutoText generates an error.

See also

AutoText Statements and Functions

AutoText

AutoTextName\$()

CountAutoTextEntries()

EditAutoText

GetAutoText\$()

InsertAutoText

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

BookmarkName\$() Example

This example puts a list of every bookmark name in a document into the array `mark$()`. You could use this array to present a list of bookmark names in a dialog box. Note that the size of the array is one less than the number of bookmarks because the subscript for the first array element is 0 (zero), not 1.

```
numBookmarks = CountBookmarks()  
arraySize = numBookmarks - 1  
Dim mark$(arraySize)  
For n = 0 To arraySize  
    mark$(n) = BookmarkName$(n + 1)  
Next
```

BookmarkName\$()

Example

BookmarkName\$(Count)

Returns the name of the bookmark specified by Count.

Argument	Explanation
Count	The number of the bookmark, from 1 to the total number of bookmarks defined for the active document (you can obtain the total using CountBookmarks()). The order of bookmark names is determined by the order of the bookmarks in the document. You must specify Count; otherwise, the function returns an error. For example, a\$ = BookmarkName\$() generates an error.

See also

Bookmarks Statements and Functions

CountBookmarks()

GetBookmark\$()

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

CmpBookmarks() Example

This example adds a string of characters in front of every line in a selection. The example first marks the selected text with a bookmark and then uses a While...Wend loop controlled by three CmpBookmarks() functions to add text in front of each line. The first CmpBookmarks() function tests whether the insertion point and the selection, stored in the "Temp" bookmark, begin at the same point; this is true when the loop begins. The second CmpBookmarks() function tests whether the insertion point is contained within "Temp"; this is true as long as the insertion point is within the original selection. The third CmpBookmarks() function tests whether the insertion point is at the end of the original selection. When the insertion point moves beyond the original selection, the loop ends. Within the While...Wend loop is yet another CmpBookmarks() instruction, which determines whether the selection is at the end of the document, a special case.

```
CopyBookmark "\Sel", "Temp"
SelType 1
While CmpBookmarks("\Sel", "Temp") = 8 \
    Or CmpBookmarks("\Sel", "Temp") = 6 \
    Or CmpBookmarks("\Sel", "Temp") = 10 \
    And leaveloop <> 1
    EndOfLine
    If CmpBookmarks("\Sel", "\EndOfDoc") = 0 Then leaveloop = 1
    StartOfLine
    Insert "****"
    LineDown
Wend
EditGoTo "Temp"
EditBookmark "Temp", .Delete
```


CmpBookmarks()

Example

CmpBookmarks(Bookmark1\$, Bookmark2\$)

Compares the contents of two bookmarks. Use CmpBookmarks() with the predefined bookmarks in Word to check the location of the insertion point or to create a macro that operates only within an area marked with a bookmark. For example, using the "\ Sel" (current selection) bookmark and the "\ Para" bookmark, you can set up a macro to operate only within a particular paragraph. For more information about predefined bookmarks, see Predefined Bookmarks.

Argument	Explanation
Bookmark1\$	The first bookmark
Bookmark2\$	The second bookmark

This function returns the following values.

Value	Explanation
0 (zero)	Bookmark1\$ and Bookmark2\$ are equivalent.
1	Bookmark1\$ is entirely below Bookmark2\$.
2	Bookmark1\$ is entirely above Bookmark2\$.
3	Bookmark1\$ is below and inside Bookmark2\$.
4	Bookmark1\$ is inside and above Bookmark2\$.
5	Bookmark1\$ encloses Bookmark2\$.
6	Bookmark2\$ encloses Bookmark1\$.
7	Bookmark1\$ and Bookmark2\$ begin at the same point, but Bookmark1\$ is longer.
8	Bookmark1\$ and Bookmark2\$ begin at the same point, but Bookmark2\$ is longer.
9	Bookmark1\$ and Bookmark2\$ end at the same place, but Bookmark1\$ is longer.
10	Bookmark1\$ and Bookmark2\$ end at the same place, but Bookmark2\$ is longer.
11	Bookmark1\$ is below and adjacent to Bookmark2\$.
12	Bookmark1\$ is above and adjacent to Bookmark2\$.
13	One or both of the bookmarks do not exist.

See also

Bookmarks Statements and Functions

CopyBookmark

EditBookmark

EmptyBookmark

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

CopyBookmark Example

This example selects the current section, then sets one bookmark at the start of the section and another bookmark at the end. You can use this technique to define starting points and end points between which your macro operates.

```
EditGoTo "\Section"  
CopyBookmark "\StartOfSel", "SectionStart"  
CopyBookmark "\EndOfSel", "SectionEnd"
```

CopyBookmark

Example

CopyBookmark Bookmark1\$, Bookmark2\$

Sets Bookmark2\$ to the insertion point or range of text marked by Bookmark1\$. You can use this statement with predefined bookmarks---such as "\ StartOfSel" and "\ EndOfSel" --- to set bookmarks relative to the insertion point or selection. For more information about predefined bookmarks, see Predefined Bookmarks.

See also

Bookmarks Statements and Functions

CmpBookmarks()

EditBookmark

SetEndOfBookmark

SetStartOfBookmark

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

CountBookmarks() Examples

This example creates an array containing every bookmark in the active document:

```
size = CountBookmarks() - 1
Dim marks$(size)
For count = 0 To size
    marks$(count) = BookmarkName$(count + 1)
Next
```

The following example deletes all the bookmarks in the active document:

```
For n = 1 To CountBookmarks()
    EditBookmark .Name = BookmarkName$(CountBookmarks()), \
        .Delete
Next
```

CountBookmarks()
CountBookmarks()

~~Example~~ Returns the number of bookmarks in the active document. As the first example in this entry demonstrates, you can use this function to define an array containing every bookmark in a document.

See also

Bookmarks Statements and Functions

BookmarkName\$()

EditBookmark

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

EditBookmark Example

This example searches for a paragraph containing only the word "Index" (that is, the heading for the index), and then, if the heading is found, adds a bookmark in front of it. You could use this bookmark in another EditBookmark instruction or with EditGoTo to move the insertion point to the index.

```
StartOfDocument
EditFind .Find = "^pIndex^p", .MatchCase = 1, \
        .Direction = 0, .Format = 0
If EditFindFound() Then
    CharLeft : CharRight
    EditBookmark .Name = "Index", .Add
End If
```

EditBookmark

Example

`EditBookmark .Name = text [, .SortBy = number] [, .Add] [, .Delete] [, .Goto]`

Adds, deletes, or selects the specified bookmark. The arguments for the EditBookmark statement correspond to the options in the Bookmark dialog box (Edit menu).

Argument	Explanation
.Name	The name of the bookmark
.SortBy	Controls how the list of bookmarks is sorted when you display the Bookmark dialog box with a Dialog or Dialog() instruction: 0 (zero) By name 1 By location

You can specify only one of the following arguments.

Argument	Explanation
.Add	Adds a bookmark at the insertion point or selection
.Delete	Deletes the bookmark
.Goto	Moves the insertion point or selection to the bookmark

If you do not specify .Add, .Delete, or .Goto, Word adds the bookmark.

See also

[Bookmarks Statements and Functions](#)

[BookmarkName\\$\(\)](#)

[CmpBookmarks\(\)](#)

[CopyBookmark](#)

[CountBookmarks\(\)](#)

[EditGoTo](#)

[EmptyBookmark\(\)](#)

[ExistingBookmark\(\)](#)

[GetBookmark\\$\(\)](#)

[SetEndOfBookmark](#)

[SetStartOfBookmark](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

EmptyBookmark() Example

This example verifies that the bookmark referred to in each REF field both exists and is not empty. If a reference to a nonexistent or empty bookmark is encountered, an appropriate message box is displayed.

```
StartOfDocument
ViewFieldCodes 1
EditFind .Find = "^d REF", .Format = 0, .Wrap = 0
While EditFindFound()
    CharLeft
    WordRight 2
    WordRight 1, 1
    mark$ = RTrim$(Selection$())
    If Not ExistingBookmark(mark$) Then
        MsgBox mark$ + " is not a bookmark."
    ElseIf EmptyBookmark(mark$) Then
        MsgBox mark$ + " is an empty bookmark."
    End If
    CharRight
    EditFind .Find = "^d REF", .Format = 0, .Wrap = 0
Wend
```


EmptyBookmark()

Example

EmptyBookmark(Name\$)

Determines whether Name\$ is an "empty" bookmark. An empty bookmark marks only a location for the insertion point in a document; it does not mark any text. You can use EmptyBookmark() to verify that a bookmark (for example, a bookmark referred to in a REF field) does indeed mark text.

This function returns the following values.

Value	Explanation
-1	If the bookmark is empty (that is, it marks no text)
0 (zero)	If the bookmark is not empty or does not exist

See also

Bookmarks Statements and Functions

BookmarkName\$()

CmpBookmarks()

CountBookmarks()

EditBookmark

ExistingBookmark()

GetBookmark\$()

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

ExistingBookmark() Example

This macro displays a prompt in the status bar for the name of a bookmark to add. If the bookmark does not yet exist, it is added. If the bookmark already exists, Word displays a message box that asks whether to reset the bookmark. If the user answers No, the macro ends. Otherwise, the bookmark is reset.

```
Sub MAIN
Input "Bookmark to add", myMark$
If ExistingBookmark(myMark$) Then
  ans = MsgBox(myMark$ + " already exists; reset?", 36)
  If ans = 0 Then Goto bye
End If
EditBookmark myMark$, .Add
bye:
End Sub
```

ExistingBookmark()

Example

ExistingBookmark(Name\$)

Indicates whether the bookmark specified by Name\$ exists in the active document. This function returns the following values.

Value	Explanation
-1	If the bookmark exists
0 (zero)	If the bookmark does not exist

See also

Bookmarks Statements and Functions

BookmarkName\$()

CmpBookmarks()

CountBookmarks()

EditBookmark

EmptyBookmark()

GetBookmark\$()

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

GetBookmark\$() Examples

This example sets the variable `first$` to the text of the first bookmark in the document:

```
first$ = GetBookmark$(BookmarkName$(1))
```

The following example sets the variable `paratext$` to the text of the paragraph containing the insertion point:

```
paratext$ = GetBookmark$("\Para")
```

The bookmark "`\Para`" is one of several predefined bookmarks that Word defines and updates automatically. For more information, see [Predefined Bookmarks](#).

GetBookmark\$()

Example

GetBookmark\$(Name\$)

Returns the text (unformatted) marked by the specified bookmark. If Name\$ is not the name of a bookmark in the active document, GetBookmark\$() returns an empty string ("").

See also

Bookmarks Statements and Functions

BookmarkName\$()

CountBookmarks()

EditBookmark

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

SetEndOfBookmark Example

This example marks the end of the current selection with the bookmark "EndPoint":

```
SetEndOfBookmark "\Sel", "EndPoint"
```

The bookmark "\ Sel" is one of several predefined bookmarks that Word defines and updates automatically. For more information, see [Predefined Bookmarks](#).

SetEndOfBookmark

Example

SetEndOfBookmark Bookmark1\$ [, Bookmark2\$]

Marks the end point of Bookmark1\$ with Bookmark2\$. If Bookmark2\$ is omitted, Bookmark1\$ is set to its own end point.

See also

Bookmarks Statements and Functions

CopyBookmark

EditBookmark

SetStartOfBookmark

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

SetStartOfBookmark Example

This example marks either end of the current paragraph with bookmarks:

```
SetStartOfBookmark "\Para", "BeginPara"  
SetEndOfBookmark "\Para", "EndPara"
```

The bookmark "\ Para" is one of several predefined bookmarks that Word defines and updates automatically. For more information, see [Predefined Bookmarks](#).

SetStartOfBookmark

Example

SetStartOfBookmark Bookmark1\$ [, Bookmark2\$]

Marks the starting point of Bookmark1\$ with Bookmark2\$. If Bookmark2\$ is omitted, Bookmark1\$ is set to its own starting point.

See also

Bookmarks Statements and Functions

CopyBookmark

EditBookmark

SetEndOfBookmark

ewc shareres, T3EWClass, \$\$button:WordBASICCclosewc shareres, T3EWClass, \$\$button:WordBASICCcopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

BorderBottom Example

This example applies a bottom border using one of two line styles, depending on whether the selection is within a table. If the selection is within a table, a double border is applied; otherwise, a thick, single border is applied.

```
If SelInfo(12) = - 1 Then
    BorderLineStyle 8
    BorderBottom 1
Else
    BorderLineStyle 4
    BorderBottom 1
End If
```

BorderBottom, BorderBottom()

Example

BorderBottom [On]

BorderBottom()

The BorderBottom statement applies or removes a bottom border for the selected paragraphs, table cells, or graphic. Note that when you apply a bottom border to a series of paragraphs or table rows, the border appears only beneath the last paragraph or row in the series. If you want a border to separate each paragraph or row, use BorderInside.

Argument	Explanation
On	Specifies whether to apply or remove a bottom border: 1 Applies the border 0 (zero) Removes the border Omitted Toggles the border

The BorderBottom() function returns the following values.

Value	Explanation
0 (zero)	If at least one of the selected items has no bottom border or if the selection contains a mixture of items (for example, a paragraph and a table cell)
1	If each item in the selection is of the same type and has a bottom border

See also

Borders and Frames Statements and Functions

BorderInside

BorderLeft

BorderLineStyle

BorderNone

BorderOutside

BorderRight

BorderTop

FormatBordersAndShading

ShadingPattern

BorderInside, BorderInside()

BorderInside [On]

BorderInside()

The BorderInside statement applies or removes inside borders for the selected paragraphs or table cells. The following illustrations show inside borders within a series of paragraphs and a table.

Lorem ipsum	Lorem ipsum	Lorem ipsum
Lorem ipsum	Lorem ipsum	Lorem ipsum
Lorem ipsum	Lorem ipsum	Lorem ipsum

Inside borders for paragraphs Inside borders for a table

The BorderInside() function returns either 0 (zero) or 1, depending on whether all the selected paragraphs or table cells are formatted with an inside border. Note that BorderInside() returns 0 (zero) if the selection is a single table cell, regardless of the borders applied to the surrounding group of cells; a single table cell can have bottom, left, right, and top borders, but not inside borders.

For complete descriptions of arguments and return values, see [BorderBottom](#).

See also

[Borders and Frames Statements and Functions](#)

[BorderBottom](#)

[BorderLeft](#)

[BorderLineStyle](#)

[BorderNone](#)

[BorderOutside](#)

[BorderRight](#)

[BorderTop](#)

[FormatBordersAndShading](#)

[ShadingPattern](#)

BorderLeft, BorderLeft()

BorderLeft [On]

BorderLeft()

The BorderLeft statement applies or removes left borders for the selected paragraphs, table cells, or graphic. The BorderLeft() function returns either 0 (zero) or 1, depending on whether the selected graphic or all the selected paragraphs or table cells are formatted with a left border.

For complete descriptions of arguments and return values, see [BorderBottom](#).

See also

[Borders and Frames Statements and Functions](#)

[BorderBottom](#)

[BorderInside](#)

[BorderLineStyle](#)

[BorderNone](#)

[BorderOutside](#)

[BorderRight](#)

[BorderTop](#)

[FormatBordersAndShading](#)

[ShadingPattern](#)

BorderStyle, BorderLineStyle()

BorderStyle Style

BorderStyle()

The BorderLineStyle statement specifies the line style for subsequent BorderBottom, BorderInside, BorderLeft, BorderOutside, BorderRight, and BorderTop instructions.

Argument	Explanation
Style	One of 12 line styles:
	0 (zero) None
	1 _____
2	_____
3	_____
4	_____
5	_____
6	_____
7	_____
8	_____
9	_____
10
11	-----

For an example that uses BorderLineStyle, see [BorderBottom Example](#).

The BorderLineStyle() function returns a number from 0 (zero) to 11 that corresponds to the line style that will be applied by subsequent border instructions. Note that this line style does not necessarily match the line style of borders in the selected paragraphs, table cells, or graphic.

See also

[Borders and Frames Statements and Functions](#)

[BorderBottom](#)

[BorderInside](#)

[BorderLeft](#)

[BorderNone](#)

[BorderOutside](#)

[BorderRight](#)

[BorderTop](#)

[FormatBordersAndShading](#)

[ShadingPattern](#)

BorderNone, BorderNone()

BorderNone [Remove]

BorderNone()

The BorderNone statement removes or applies all borders (left, right, top, bottom, and inside) for the selected items. You can remove or apply all borders for a series of paragraphs or table rows, but not a combination of paragraphs and table rows. To remove or apply borders for a graphic, you must first select only that graphic.

Argument	Explanation
Remove	Specifies whether to remove or apply all borders for the selection: 0 (zero) Applies borders 1 or omitted Removes borders

The BorderNone() function returns 0 (zero) if the selection contains at least one border and 1 if the selection contains no borders.

See also

[Borders and Frames Statements and Functions](#)

[BorderBottom](#)

[BorderInside](#)

[BorderLeft](#)

[BorderLineStyle](#)

[BorderOutside](#)

[BorderRight](#)

[BorderTop](#)

[FormatBordersAndShading](#)

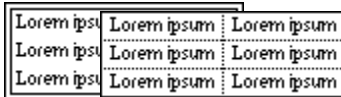
[ShadingPattern](#)

BorderOutside, BorderOutside()

BorderOutside [On]

BorderOutside()

The BorderOutside statement applies or removes outside borders for the selected paragraphs, table cells, or graphic. The following illustrations show outside borders applied to a series of paragraphs and an entire table.



Lorem ipsum	Lorem ipsum	Lorem ipsum
Lorem ipsum	Lorem ipsum	Lorem ipsum
Lorem ipsum	Lorem ipsum	Lorem ipsum

Outside borders for paragraphs

Outside borders for a table

The BorderOutside() function returns either 0 (zero) or 1, depending on whether the selected graphic or all the selected paragraphs or table cells are formatted with an outside border.

For complete descriptions of arguments and return values, see [BorderBottom](#).

See also

[Borders and Frames Statements and Functions](#)

[BorderBottom](#)

[BorderInside](#)

[BorderLeft](#)

[BorderLineStyle](#)

[BorderNone](#)

[BorderRight](#)

[BorderTop](#)

[FormatBordersAndShading](#)

[ShadingPattern](#)

BorderRight, BorderRight()

BorderRight [On]

BorderRight()

The BorderRight statement applies or removes right borders for the selected paragraphs, table cells, or graphic. The BorderRight() function returns either 0 (zero) or 1, depending on whether the selected graphic or all the selected paragraphs or table cells are formatted with a right border.

For complete descriptions of arguments and return values, see [BorderBottom](#).

See also

[Borders and Frames Statements and Functions](#)

[BorderBottom](#)

[BorderInside](#)

[BorderLeft](#)

[BorderLineStyle](#)

[BorderNone](#)

[BorderOutside](#)

[BorderTop](#)

[FormatBordersAndShading](#)

[ShadingPattern](#)

BorderTop, BorderTop()

BorderTop [On]

BorderTop()

The BorderTop statement applies or removes a top border for the selected paragraphs, table cells, or graphic. Note that when you apply a top border to a series of paragraphs or table rows, the border appears only above the first paragraph or row in the series. If you want a border to separate each paragraph or row, use BorderInside.

The BorderTop() function returns either 0 (zero) or 1, depending on whether the selected graphic or all the selected paragraphs or table cells are formatted with a top border.

For complete descriptions of arguments and return values, see [BorderBottom](#).

See also

[Borders and Frames Statements and Functions](#)

[BorderBottom](#)

[BorderInside](#)

[BorderLeft](#)

[BorderLineStyle](#)

[BorderNone](#)

[BorderOutside](#)

[BorderRight](#)

[FormatBordersAndShading](#)

[ShadingPattern](#)

FormatBordersAndShading

FormatBordersAndShading [.ApplyTo = number] [, .Shadow = number] [, .TopBorder = number] [, .LeftBorder = number] [, .BottomBorder = number] [, .RightBorder = number] [, .HorizBorder = number] [, .VertBorder = number] [, .TopColor = number] [, .LeftColor = number] [, .BottomColor = number] [, .RightColor = number] [, .HorizColor = number] [, .VertColor = number] [, .FineShading = number] [, .FromText = number or text] [, .Shading = number] [, .Foreground = number] [, .Background = number] [, .Tab = text]

Sets border and shading formats for the selected paragraphs, table cells, or graphic. The arguments for the FormatBordersAndShading statement correspond to the options in the Borders And Shading dialog box (Format menu).

Argument	Explanation
.ApplyTo	<p>If the selection consists of more than one of the following items, specifies to which item or items the border format is applied:</p> <ul style="list-style-type: none">0 (zero) Paragraphs1 Graphic2 Cells3 Whole table <p>If .ApplyTo is omitted, the default for the selection is assumed.</p>
.Shadow	<p>Specifies whether to apply a shadow to the border of paragraphs or a graphic:</p> <ul style="list-style-type: none">0 (zero) Does not apply a shadow.1 Applies a shadow. <p>You cannot apply a shadow to a table or table cells. If you want to apply a shadow to a paragraph or graphic, the item must have---or you must specify---matching right, left, top, and bottom borders. Otherwise, an error occurs.</p>
.TopBorder, .LeftBorder, .BottomBorder, .RightBorder	<p>The line style for the border on the top, left, bottom, and right edges of paragraphs, cells, or a graphic, in the range 0 (zero), which is no border, through 11 (for a list of line styles and their values, see BorderLineStyle).</p>
.HorizBorder	<p>The line style for the horizontal border between paragraphs or table cells, in the range 0 (zero), which is no border, through 11. The border does not appear unless it is applied to at least two consecutive paragraphs or table rows.</p>
.VertBorder	<p>The line style for the vertical border between table cells, in the range 0 (zero), which is no border, through 11. The border does not appear unless the table selection is at least two cells wide. (When applied to paragraphs, .VertBorder has the same effect as .LeftBorder.)</p>
.TopColor, .LeftColor, .BottomColor, .RightColor, .HorizColor, .VertColor	<p>The color to be applied to the specified borders, in the range from 0 (zero), which is Auto, through 16 (for a list of colors and their values, see CharColor).</p>

<code>.FineShading</code>	A shading pattern in the range 0 (zero) to 40 corresponding to a shading percentage in 2.5 percent increments. If <code>.FineShading</code> is anything but 0 (zero), <code>.Shading</code> is ignored.
<code>.FromText</code>	The distance of the border from adjacent text, in points or a text measurement. Valid only for paragraphs; otherwise, <code>.FromText</code> must be an empty string (" ") or omitted or an error will occur.
<code>.Shading</code>	The shading pattern to be applied to the selection, in the range from 0 (zero), which is Clear, through 25 (for a list of shading patterns and their values, see ShadingPattern).
<code>.Foreground</code>	The color to be applied to the foreground of the shading, in the range from 0 (zero), which is Auto, through 16 (for a list of colors and their values, see CharColor).
<code>.Background</code>	The color to be applied to the background of the shading, in the range from 0 (zero), which is Auto, through 16.
<code>.Tab</code>	Specifies which tab to select when you display the Borders And Shading dialog box with a <code>Dialog</code> or <code>Dialog()</code> instruction: 0 (zero) Borders tab 1 Shading tab

See also

[Borders and Frames Statements and Functions](#)

[Border Top](#)

[BorderBottom](#)

[BorderInside](#)

[BorderLeft](#)

[BorderLineStyle](#)

[BorderNone](#)

[BorderOutside](#)

[BorderRight](#)

[ShadingPattern](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCclosewc shareres, T3EWClass, \$\$button:WordBASICCcopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

FormatFrame Example

This example selects and frames the current paragraph and then formats the frame as left-aligned, relative to the current column, with a 0.13-inch gap between the frame and text above and below:

```
EditGoTo "\Para"
```

```
InsertFrame
```

```
FormatFrame .PositionHorz = 0, .PositionHorzRel = 2, \  
  .DistVertFromText = "0.13 in"
```

FormatFrame

Example

```
FormatFrame [.Wrap = number] [, .WidthRule = number] [, .FixedWidth = number or text] [, .  
HeightRule = number] [, .FixedHeight = number or text] [, .PositionHorz = number or text] [, .  
PositionHorzRel = number] [, .DistFromText = number or text] [, .PositionVert = number or text] [, .  
PositionVertRel = number] [, .DistVertFromText = number or text] [, .MoveWithText = number] [, .  
LockAnchor = number] [, .RemoveFrame]
```

Positions and sets options for the selected frame. If the insertion point or selection is not within a frame, an error occurs. The arguments for the FormatFrame statement correspond to the options in the Frame dialog box (Format menu).

Argument	Explanation
.Wrap	Specifies a Text Wrapping option: 0 (zero) Text does not wrap around the frame. 1 Text wraps around the frame.
.WidthRule	The rule used to determine the width of the frame: 0 (zero) Auto (determined by paragraph width). 1 Exactly (width will be exactly . FixedWidth).
.FixedWidth	If .WidthRule is 1, the width of the frame in points or a text measurement.
.HeightRule	The rule used to determine the height of the frame: 0 (zero) Auto (determined by paragraph height). 1 At Least (height will be no less than . FixedHeight). 2 Exactly (height will be exactly . FixedHeight).
.FixedHeight	If .HeightRule is 1 or 2, the height of the frame in points or a text measurement (1 inch = 72 points).
.PositionHorz	Horizontal distance, in points or a text measurement, from the edge of the item specified by . PositionHorzRel. You can also specify "Left," "Right," "Center," "Inside," and "Outside" as text arguments.
.PositionHorzRel	Specifies that the horizontal position is relative to: 0 (zero) Margin 1 Page 2 Column
.DistFromText	Distance between the frame and the text to its left, right, or both, in points or a text measurement.
.PositionVert	Vertical distance, in points or a text measurement, from the edge of the item specified by . PositionVertRel. You can also specify "Top," "Bottom," and "Center" as text arguments.
.PositionVertRel	Specifies that the vertical position is relative to: 0 (zero) Margin 1 Page 2 Paragraph
.DistVertFromText	Distance between the frame and the text above, below, or both, in points or a text measurement.

<code>.MoveWithText</code>	If 1, the frame moves as text is added or removed around it.
<code>.LockAnchor</code>	If 1, the frame anchor (which indicates where the frame will appear in normal view) remains fixed when the associated frame is repositioned. A locked frame anchor cannot be repositioned.
<code>.RemoveFrame</code>	Removes the frame format from the selected text or graphic.

See also

Borders and Frames Statements and Functions

FormatDefineStyleFrame

InsertFrame

RemoveFrames

ewc shareres, T3EWClass, \$\$button:WordBASICCclosewc shareres, T3EWClass, \$\$button:WordBASICCcopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

InsertFrame Example

This example inserts a frame and then positions it in the margin to the left of the current paragraph, so the user can type a margin note in it. If the active document is not in page layout view, Word displays a message box asking if the user wants to switch to page layout view.

```
SelType 1
If ViewPage() = 0 Then
    ans = MsgBox("Switch to page layout view?", \
        "Insert Margin Note", 36)
    If ans = - 1 Then ViewPage
End If
InsertFrame
FormatFrame .Wrap = 1, .WidthRule = 1, .FixedWidth = ".75 in", \
    .PositionHorz = "Left", .PositionHorzRel = 1, \
    .DistFromText = "0.13 in", .PositionVert = "0", \
    .PositionVertRel = 2, .DistVertFromText = "0"
SelType 1 : FontSize 8 : Italic 1
HScroll 0
```

InsertFrame

Example

InsertFrame

Inserts an empty frame, or frames the selected text, graphic, or both. If there is no selection, Word inserts a 1-inch - square frame at the insertion point (the frame appears as a square in page layout view). You can change the dimensions of the frame with FormatFrame.

See also

Borders and Frames Statements and Functions

FormatFrame

RemoveFrames

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

RemoveFrames Example

This example removes all frames from the entire document:

```
EditSelectAll
```

```
RemoveFrames
```

RemoveFrames

Example

RemoveFrames

Removes all frames in the selection. Note that borders, applied automatically when you insert a frame around text, are not removed.

See also

Borders and Frames Statements and Functions

FormatBordersAndShading

FormatFrame














InsertFrame

ShadingPattern, ShadingPattern()

ShadingPattern Type

ShadingPattern()

The ShadingPattern statement applies one of 26 shading formats to the selected paragraphs, table cells, or frame.

Argument	Explanation
Type	The shading format to apply:
	0 
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

The ShadingPattern() function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is shaded (the shading pattern is Clear)
-1	If the selection contains a mixture of shading patterns
1 through 25	If all the selection is formatted with the same shading pattern

See also

[Borders and Frames Statements and Functions](#)

[FormatBordersAndShading](#)

ViewBordersToolBar

Displays the Borders toolbar if it is hidden or hides the Borders toolbar if it is displayed.

See also

Borders and Frames Statements and Functions

ViewDrawingToolBar

ViewToolbars

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

Call Example

This example calls the subroutine `FindName` twice; each line, with or without `Call`, has the same effect:

```
Call FindName      'Transfer control to the subroutine FindName  
FindName          'Transfer control to the subroutine FindName
```


Call

Example

`[Call] [MacroName][.][SubName] [ArgumentList]`

Transfers control to a subroutine in the running macro or another macro. To specify a subroutine in another macro, use the syntax `MacroName.SubName`. If `SubName` is not specified, the `Main` subroutine in `MacroName` runs. `Call` is optional; it can help distinguish subroutine names from WordBasic keywords when you read and edit macros. Each variable in the comma-delimited `ArgumentList` must correspond to a value that the subroutine being called is prepared to receive.

Note

When you call another macro, Word looks for the macro in available templates in the following order: the template containing the `Call` instruction, the active template, the Normal template, and loaded global templates. For example, suppose `USER.DOT` and `NORMAL.DOT` both contain a `DisplayMessage` macro. The following macro in `USER.DOT`:

```
FileNew .Template = "Normal"  
DisplayMessage
```

runs the `DisplayMessage` macro in `USER.DOT`, even though a document based on `NORMAL.DOT` is active when the `Call` instruction is run.

For more information about using subroutines, including how to share variables and pass arguments between subroutines, see Chapter 4, "Advanced WordBasic," in the [Microsoft Word Developer's Kit](#).

See also

[Branching and Control Statements and Functions](#)

[Sub...End Sub](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

For...Next Examples

This example displays five message boxes in a row, each giving the current value of `count`:

```
For count = 1 To 5
    MsgBox "Current value of count is" + Str$(count)
Next count
```

The following example produces exactly the same effect as the previous example by decrementing the value of `count` in steps of -1:

```
For count = 5 To 1 Step -1
    MsgBox "Current value of count is" + Str$(count)
Next
```

The following example demonstrates how you can use WordBasic counting functions with a For...Next loop to perform an operation on all the items in a certain category. In this example, the names of all the bookmarks defined in the active document are stored in the array `mark$()`.

```
numBookmarks = CountBookmarks()
arraySize = numBookmarks - 1
Dim mark$(arraySize)
For n = 0 To arraySize
    mark$(n) = BookmarkName$(n + 1)
Next
```

For...Next

Example

For CounterVariable = Start To End [Step Increment]

Series of instructions

Next [CounterVariable]

Repeats the series of instructions between For and Next while increasing CounterVariable by 1 (default) or Increment until CounterVariable is greater than End. If Start is greater than End, Increment must be a negative value; CounterVariable decreases by Increment until it is less than End.

If you place one or more For...Next loops within another, use a unique CounterVariable for each loop, as in the following instructions:

```
For I = 1 To 10
  For J = 1 To 10
    For K = 1 To 10
      'Series of instructions
    Next K
  Next J
Next I
```

See also

Branching and Control Statements and Functions

Goto

If...Then...Else

Select Case

While...Wend

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

Function...End Function Example

This macro prompts the user to type a number of degrees Fahrenheit, which is passed to the `ConvertTemp` (`()`) function through the variable `fahrenheit`. The function converts `fahrenheit` to degrees Celsius, and then the main subroutine displays this value in a message box.

```
Sub MAIN
    On Error Resume Next
    tmp$ = InputBox$("Type a Fahrenheit temperature:")
    fahrenheit = Val(tmp$)
    celsius = ConvertTemp(fahrenheit)
    MsgBox tmp$ + " Fahrenheit =" + Str$(celsius) + " Celsius"
End Sub
```

```
Function ConvertTemp(fahrenheit)
    tmp = fahrenheit
    tmp = ((tmp - 32) * 5) / 9
    tmp = Int(tmp)
    ConvertTemp = tmp
End Function
```

Function...End Function

Example

```
Function FunctionName[$][(ArgumentList)]
```

Series of instructions to determine a value

```
FunctionName[$]
```

```
= value
```

```
End Function
```

Defines a function---a series of instructions that returns a single value. To return a string value, the function name must end with a dollar sign (\$). Note that unlike the names of built-in WordBasic functions, the names of user-defined functions that do not specify ArgumentList do not end with empty parentheses; if you include empty parentheses, an error will occur.

ArgumentList is a list of variables, separated by commas, that are passed to the function by the statement calling the function. String variables must end with a dollar sign. ArgumentList cannot include values; constants should be declared as variables and passed to the function through variable names.

See also

Branching and Control Statements and Functions

Sub...End Sub

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

Goto Example

This macro displays a message box, with Yes, No, and Cancel buttons, asking if the user wants to continue the macro. If the user chooses No or Cancel, the macro branches to the label `bye` immediately before `End Sub`, and the macro ends.

```
Sub MAIN
ans = MsgBox("Continue macro?", 3)
If ans = 0 Or ans = 1 Then Goto bye
'Series of instructions to run if the user chooses Yes
bye:
End Sub
```

Goto

Example

Goto Label

Redirects a running macro from the Goto instruction to the specified Label anywhere in the same subroutine or function. The macro continues running from the instruction that follows the label. Keep the following in mind when placing a label in a macro:

- Labels must be the first text on a line and cannot be preceded by spaces or tab characters.
- Labels must be followed by a colon (:). (Do not include the colon in the Goto instruction.)
- Labels that contain letters must begin with a letter and can contain letters and numbers up to a maximum length of 40 characters, not counting the colon.
- You can use a number that appears at the beginning of a line instead of a label. Line numbers are supported primarily for compatibility with Basic programs created in older versions of the Basic programming language that require line numbers. The line number can be as high as 32759 and does not need a colon following it.

See also

Branching and Control Statements and Functions

For...Next

If...Then...Else

Select Case

While...Wend

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

If...Then...Else Examples

This example applies bold formatting to the entire selection if the selection is partially bold:

```
If Bold() = -1 Then Bold 1
```

The following example applies italic formatting if the selection is entirely bold; otherwise, underline formatting is applied:

```
If Bold() = 1 Then Italic 1 Else Underline 1
```

The following example shows how you can use a compound expression as the condition (in this case, whether the selection is both bold and italic):

```
If Bold() = 1 And Italic() = 1 Then ResetChar
```

The following example uses the full syntax available with the If conditional. The conditional could be described as follows: "If the selection is entirely bold, make it italic. If the selection is partially bold, reset the character formatting. Otherwise, make the selection bold."

```
If Bold() = 1 Then  
    Italic 1  
ElseIf Bold() = -1 Then  
    ResetChar  
Else  
    Bold 1  
End If
```


If...Then...Else

Example

```
If Condition Then Instruction [Else Instruction]
```

```
If Condition1 Then
```

```
    Series of instructions
```

```
[ElseIf Condition2 Then
```

```
    Series of instructions]
```

```
[Else
```

```
of instructions]
```

```
End If
```

Series

Runs instructions conditionally. In the simplest form of the If conditional --- If Condition Then Instruction --- the Instruction runs if Condition is true. In WordBasic, "true" means the condition evaluates to -1 and "false" means the condition evaluates to 0 (zero).

You can write an entire If conditional on one line if you specify one condition following If and one instruction following Then (and one instruction following Else, if included). Do not conclude this form of the conditional with End If. Note that it is possible to specify multiple instructions using this form if you separate the instructions with colons, as in the following conditional:

```
If Bold() = 1 Then Bold 0 : Italic 1
```

In general, if you need to specify a series of conditional instructions, the full syntax is preferable to separating instructions with colons. With the full syntax, you can use ElseIf to include a second condition nested within the If conditional. You can add as many ElseIf instructions to an If conditional as you need.

See also

[Branching and Control Statements and Functions](#)

[For...Next](#)

[Goto](#)

[Select Case](#)

[While...Wend](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

On Error Examples

This example shows a common use of On Error Resume Next to avoid WordBasic error number 102, "Command failed," when a user cancels a dialog box or prompt:

```
On Error Resume Next
A$ = InputBox$("Your name please:")
```

The following macro prompts the user to specify a sequential file for input (for example, a text-only file containing a list of Word documents). If the file cannot be found, the instructions following the label specified by On Error Goto Label suggest a reason corresponding to the error number.

```
Sub MAIN
On Error Goto ErrorHandler
DocName$ = InputBox$("Filename for input:", "", DocName$)
Open DocName$ For Input As #1
'Statements that use the input go here
Close #1
Goto Done          'If there is no error, skip the error handler
ErrorHandler:
Select Case Err
  Case 53 : MsgBox "The file " + DocName$ + " does not exist."
  Case 64 : MsgBox "The specified drive is not available."
  Case 76 : MsgBox "The specified directory does not exist."
  Case 102          'If the user cancels the dialog box
  Case Else : MsgBox "Error" + Str$(Err) + " occurred."
End Select
Err = 0
Done:
End Sub
```

On Error

Example

On Error Goto Label

On Error Resume Next

On Error Goto 0

Establishes an "error handler" --- typically, a series of instructions that takes over when an error occurs. When an error occurs in a macro that does not contain the On Error statement, an error message is displayed and the macro quits.

This form

Performs this action

On Error Goto Label

Jumps from the line where the error occurred to the specified label. The instructions following this label can then determine the nature of the error (using the special variable Err) and take some appropriate action to correct or resolve the problem. For more information, see Err.

On Error Resume Next

Continues running the macro from the line that follows the line where the error occurred and resets Err to 0 (zero). In effect, the error is ignored.

On Error Goto 0

Disables the error trapping established by an earlier On Error Goto or On Error Resume Next statement and sets Err to 0 (zero).

Once an error triggers an error handler, no further error handling occurs until Err is reset to 0 (zero).

Usually, you should place an `Err = 0` instruction at the end of your error handler. Do not include `Err = 0` in the middle of an error handler or you risk creating an endless loop if an error occurs within the handler.

Note that an error handler established in the main subroutine is not in effect when control passes to another subroutine. To trap all errors, each subroutine must have its own On Error statement and error handler.

After control is returned to the main subroutine, the main On Error instruction is again in effect.

WordBasic generates errors with numbers less than 1000; Word itself generates errors with numbers 1000 or greater. Error handlers can trap both WordBasic and Word errors. However, if a Word error occurs, an error message is displayed, and the user must respond before the macro can continue. When the user chooses the OK button, control passes to the error handler.

For a complete list of all WordBasic and Word error messages and error numbers, see Error Messages.

See also

Branching and Control Statements and Functions

Err

Error

Goto

Select Case

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

Select Case Examples

This example goes to each paragraph in the document and inserts either a bullet or a hyphen, depending on whether the paragraph's style is "ListItem1" or "ListItem2." If a paragraph that is not formatted with either of these styles is found, the instruction following Case Else displays a message box.

```
StartOfDocument
While CmpBookmarks("\Sel", "\EndOfDoc") <> 0
  Select Case StyleName$()
    Case "ListItem1"
      ToolsBulletsNumbers .Type = 0
    Case "ListItem2"
      Insert "-" + Chr$(9)
    Case Else
      MsgBox "Not a list style"
  End Select
  ParaDown
Wend
```

The following example illustrates how Select Case may be used to evaluate numeric expressions. The Select Case instruction generates a random number between -5 and 5, and the subsequent Case instructions run depending on the value of that numeric expression.

```
Select Case Int(Rnd() * 10) - 5
  Case 1,3
    Print "One or three"
  Case Is > 3
    Print "Greater than three"
  Case -5 To 0
    Print "Between -5 and 0 (inclusive)"
  Case Else
    Print "Must be 2"
End Select
```

Select Case

Example

Select Case Expression

Case CaseExpression

Series of instruction(s)

[Case Else

Series

of instruction(s)]

End Select

Runs one of several series of instructions according to the value of Expression. Expression is compared with each CaseExpression in turn. When a match is found, the instructions following that Case CaseExpression are run, and then control passes to the instruction following End Select. If there is no match, the instructions following Case Else are run. If there is no match and there is no Case Else instruction, an error occurs.

The Select Case control structure is an important part of most dialog functions. For more information about dialog functions, see Chapter 5, "Working with Custom Dialog Boxes," in the Microsoft Word Developer's Kit.

Keep the following points in mind when using Select Case:

- Use the Is keyword to compare CaseExpression with Expression using a relational operator. For example, the instruction `Case Is > 8` tests for any value greater than 8. Do not use the Is keyword without a relational operator or an error will occur; for example, `Case Is 8` generates an error.
- Use the To keyword to test for a value that falls within a specified range. For example, the instruction `Case 4 To 8` tests for any value greater than or equal to 4 and less than or equal to 8.
- If you include a Goto instruction to go to a label outside the Select Case control structure, an error will occur.

See also

Branching and Control Statements and Functions

For...Next

Goto

If...Then...Else

While...Wend

Stop

Stop [SuppressMessage]

Stops a running macro. If SuppressMessage is -1, no message appears. Otherwise, Word displays a message box that says the macro was interrupted. When Word encounters a Stop instruction in a macro that is open in a macro-editing window, you can click the Continue button on the Macro toolbar to continue running the macro.

See also

Branching and Control Statements and Functions

ShowVars

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

Sub...End Sub Example

In this macro, the main subroutine calls the `GoBeep` subroutine, passing the number of times to beep through the variable `numBeeps`:

```
Sub MAIN
    numBeeps = 3
    GoBeep(numBeeps)
End Sub

Sub GoBeep(count)
    For n = 1 To count
        Beep
        For t = 1 To 100 : Next           'Add time between beeps
    Next
End Sub
```

If the `GoBeep` subroutine were in a macro named `LibMacros`, the call to the subroutine would be as follows:

```
Sub MAIN
    numBeeps = 3
    LibMacros.GoBeep(numBeeps)
End Sub
```

For more information about using subroutines in different macros, see Chapter 4, "Advanced WordBasic," in the Microsoft Word Developer's Kit.

Sub...End Sub

Example

```
Sub SubName[(ArgumentList)]  
    Series of instructions
```

End Sub

Defines a subroutine. A subroutine is a series of instructions that can be called repeatedly from the main subroutine and can make your macros shorter and easier to debug.

Argument	Explanation
SubName	The name of the subroutine.
ArgumentList	A list of arguments, separated by commas. You can then use these arguments in the subroutine. Values, string and numeric variables, and array variables are all valid arguments.

Subroutines must appear outside the main subroutine --- generally, you add subroutines after the End Sub instruction that ends the main subroutine. You can call a subroutine not only from the macro's main subroutine, but also from other subroutines and even other macros. For more information about using subroutines, including how to share variables and pass arguments between subroutines, see Chapter 4, "Advanced WordBasic," in the Microsoft Word Developer's Kit.

See also

Branching and Control Statements and Functions

Call

Function...End Function

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

While...Wend Example

This example uses the Files\$() function within a While...Wend loop to insert a list of files in the current directory whose filenames end with the .DOC filename extension. The instruction a\$ = Files\$(*.DOC) returns the first filename with a .DOC extension and a\$ = Files\$() returns the next filename with a .DOC extension each time the instructions within the loop run. As soon as Files\$() returns an empty string (" "), indicating there are no other .DOC files in the current directory, the condition a\$ <> " " is false and Word exits the While...Wend loop.

```
FileNewDefault
currdir$ = Files$(".")
a$ = Files$ (*.DOC)
InsertPara : Insert a$
count = 1
While a$ <> " "
    count = count + 1
    a$ = Files$()
    InsertPara : Insert a$
Wend
StartOfDocument : Bold 1
Insert currdir$ + "\*.DOC: " + Str$(count - 1) + " files"
```

While...Wend

Example

While Condition1

Series of instructions

Wend

Repeats a series of instructions between **While** and **Wend** while the specified condition is true. The **While...Wend** control structure is often used in WordBasic to repeat a series of instructions each time a given piece of text or formatting is found in a Word document. For an example of this use of **While...Wend**, see EditFind.

See also

Branching and Control Statements and Functions

For...Next

Goto

If...Then...Else

Select Case

DemoteList

Demotes the selected paragraphs by one level in a multilevel list. If the selected paragraphs are formatted as a bulleted list or as a numbered list that isn't multilevel, DemoteList increases the indent. If the selected paragraphs are not already formatted as a numbered or bulleted list, an error occurs.

See also

[Bullets and Numbering Statements and Functions](#)

[FormatBulletsAndNumbering](#)

[PromoteList](#)

FormatBulletDefault, FormatBulletDefault()

FormatBulletDefault [Add]

FormatBulletDefault()

The FormatBulletDefault statement adds bullets to or removes bullets from the selected paragraphs.

Argument	Explanation
Add	Specifies whether to add or remove bullets: 0 (zero) Removes bullets. If the paragraph preceding or following the selection is not formatted as a list paragraph, the list format in the selection is removed along with the bullets. 1 Adds bullets. If the paragraph preceding the selection already has bullets applied with the Bullets And Numbering command (Format menu), the selected paragraphs are formatted with matching bullets; otherwise, the default settings of the Bullets And Numbering dialog box (Format menu) are used. Omitted Toggles bullets.

The FormatBulletDefault() function returns the following values.

Value	Explanation
0 (zero)	If none of the selected paragraphs are bulleted or numbered
-1	If the selected paragraphs are not all bulleted, all "skipped," or all formatted with the same level of numbering
1	If all the selected paragraphs are bulleted

See also

[Bullets and Numbering Statements and Functions](#)

[FormatBulletsAndNumbering](#)

[FormatNumberDefault](#)

[SkipNumbering](#)

ewc shareres, T3EWClass, \$\$button:WordBASICCclosewc shareres, T3EWClass, \$\$button:WordBASICCcopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

FormatBulletsAndNumbering Example

This example adds diamond-shaped bullets to the selected paragraphs and formats the paragraphs with a hanging indent:

```
FormatBulletsAndNumbering .Hang = 1, .Preset = 3
```

FormatBulletsAndNumbering

Example

`FormatBulletsAndNumbering [.Remove] [, .Hang = number] [, .Preset = number]`

Adds bullets or numbers to the selected paragraphs based on the preset bullets or numbering scheme you specify, or removes bullets and numbers. The arguments for the `FormatBulletsAndNumbering` statement correspond to the options in the Bullets And Numbering dialog box (Format menu). You cannot display this dialog box using a `Dialog` or `Dialog()` instruction.

Argument	Explanation
<code>.Remove</code>	Removes bullets or numbering from the selection.
<code>.Hang</code>	If 1, applies a hanging indent to the selected paragraphs.
<code>.Preset</code>	A number corresponding to a bullets or numbering scheme in the Bullets And Numbering dialog box (Format menu). To determine the appropriate number, display the Bullets And Numbering dialog box and then select the tab with the scheme you want. Counting left to right, values for the preset schemes are:

- 1 through 6 for the schemes on the Bulleted tab.
- 7 through 12 for the schemes on the Numbered tab.
- 13 through 18 for the schemes on the Multilevel tab.

See also

[Bullets and Numbering Statements and Functions](#)

[FormatBulletDefault](#)

[FormatNumberDefault](#)

[RemoveBulletsNumbers](#)

[SkipNumbering](#)

FormatNumberDefault, FormatNumberDefault()

FormatNumberDefault [On]

FormatNumberDefault()

The FormatNumberDefault statement adds numbers to or removes numbers from the selected paragraphs.

Argument	Explanation
On	Specifies whether to add or remove numbers: <ul style="list-style-type: none">0 (zero) Removes numbers. If the paragraph preceding or following the selection is not formatted as a list paragraph, the list format in the selection is removed along with the numbers.1 Adds numbers. If the paragraph preceding or following the selection already has numbers applied with the Bullets And Numbering command (Format menu), the selected paragraphs are formatted with the same numbering scheme; otherwise, the default settings of the Bullets And Numbering dialog box are used.
Omitted	Toggles numbers.

The FormatNumberDefault() function returns the following values.

Value	Explanation
0 (zero)	If none of the selected paragraphs are numbered or bulleted
-1	If the selected paragraphs are not all bulleted, all "skipped," or all formatted with the same level of numbering
1-9	If all the selected paragraphs are numbered with the same level of numbering in a multilevel list
10	If all the selected paragraphs are numbered with one of the six schemes on the Numbered tab in the Bullets And Numbering dialog box
11	If all the selected paragraphs are bulleted
12	If all the selected paragraphs are "skipped"

See also

[Bullets and Numbering Statements and Functions](#)

[FormatBulletDefault](#)

[FormatBulletsAndNumbering](#)

[SkipNumbering](#)

PromoteList

Promotes the selected paragraphs by one level in a multilevel list. If the selected paragraphs are formatted as a bulleted list or as a numbered list that isn't multilevel, PromoteList decreases the indent. If the selected paragraphs are not already formatted as a numbered or bulleted list, an error occurs.

See also

Bullets and Numbering Statements and Functions

DemoteList

FormatBulletsAndNumbering

RemoveBulletsNumbers

Removes bullets or numbers as well as list formatting from the selected paragraphs in a bulleted or numbered list created with the Bullets And Numbering command (Format menu). Subsequent bulleted or numbered paragraphs start a new list and restart the numbering in the case of a numbered list. RemoveBulletsNumbers corresponds to the Remove button in the Bullets And Numbering dialog box (Format menu).

See also

Bullets and Numbering Statements and Functions

FormatBulletsAndNumbering

SkipNumbering

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

SkipNumbering Example

This example selects the current paragraph and uses SkipNumbering() to determine whether the paragraph is skipped. If it is, numbering is reapplied to the paragraph.

```
EditGoTo "\Para"  
If SkipNumbering() = 1 Then  
    FormatBulletsAndNumbering .Hang = 1, .Preset = 8  
End If
```

SkipNumbering, SkipNumbering()
The SkipNumbering statement skips bullets or numbers for the selected paragraphs in a bulleted or numbered list created with the Bullets And Numbering command (Format menu). Subsequent bulleted or numbered paragraphs continue the current list, rather than starting a new list (and restarting the numbering in the case of a numbered list).

The SkipNumbering() function returns the following values.

Value	Explanation
0 (zero)	If the selected paragraphs are not skipped. The selected paragraphs may or may not be part of a bulleted or numbered list.
-1	If some of the selected paragraphs are skipped and some are not, or the selection includes more than one level in a multilevel list.
1	If all the selected paragraphs are skipped.

See also

[Bullets and Numbering Statements and Functions](#)

[DemoteList](#)

[FormatBulletsAndNumbering](#)

[PromoteList](#)

[RemoveBulletsNumbers](#)

ToolsBulletListDefault

ToolsBulletListDefault

Adds bullets and tab characters to the selected paragraphs and formats the paragraphs with a hanging indent. Bullets are inserted as SYMBOL fields.

Note

The ToolsBulletListDefault statement corresponds to the Bulleted List button on the Toolbar in Word version 2.x. In Word version 6.0, the Bullets button is on the Formatting toolbar and its corresponding WordBasic statement is FormatBulletDefault.

See also

Bullets and Numbering Statements and Functions

FormatBulletDefault

FormatBulletsAndNumbering

FormatNumberDefault

ToolsBulletsNumbers

ToolsNumberListDefault

ewc shareres, T3EWClass, \$\$button:WordBASICCloseewc shareres, T3EWClass, \$\$button:WordBASICCopyewc shareres,
T3EWClass, \$\$button:WordBASICPrintewc shareres, T3EWCLASS, \$\$+button:WordBASICGreyBar

ToolsBulletsNumbers Example

This example formats the selection as a bulleted list, with the bullet defined as character code 183 in the Symbol font, at 10 points in size:

```
ToolsBulletsNumbers .Font = "Symbol", .CharNum = "183", .Type = 0, \  
.Points = 10, .Hang = 1, .Indent = "0.25 in", .Replace = 0
```

ToolsBulletsNumbers

Example

ToolsBulletsNumbers [.Replace = number] [, .Font = text] [, .CharNum = text] [, .Type = number] [, .FormatOutline = text] [, .AutoUpdate = number] [, .FormatNumber = number] [, .Punctuation = text] [, .StartAt = text] [, .Points = number or text] [, .Hang = number] [, .Indent = number or text] [, .Remove]

Sets formats for bulleted, numbered, and outline-numbered paragraphs. This statement is included for compatibility with the previous version of Word; the arguments for ToolsBulletsNumbers correspond to the options in the Bullets And Numbering dialog box (Tools menu) in Word version 2.x. Not every argument applies to each type of list.

Argument	Explanation
.Replace	If 1, Word updates bullets only for paragraphs that are already bulleted, or updates numbers only for paragraphs that are already numbered.
.Font	The font for the numbers or the bullet character in a list.
.CharNum	The character or ANSI code for the character to use as the bullet. Bullets are inserted as SYMBOL fields.
.Type	The type of list to create: 0 (zero) Bulleted list 1 Numbered list 2 Outline-numbered list
.FormatOutline	A format for numbering outlines. The available formats are Legal, Outline, Sequence, Learn, and Outline All. The Learn format applies a format based on the first number for each level in the selection.
.AutoUpdate	If 1, numbers are inserted as fields that update automatically when the sequence of paragraphs changes.
.FormatNumber	Specifies a format for numbering lists: 0 (zero) 1, 2, 3, 4 1 I, II, III, IV 2 i, ii, iii, iv 3 A, B, C, D 4 a, b, c, d
.Punctuation	The separator character or characters for numbers in a list. If you specify one character, it follows each number; if you specify two characters, they enclose each number.
.StartAt	The starting number or letter for the list.
.Points	The size of the bullet character, in points, in a bulleted list.
.Hang	If 1, sets a hanging indent for the list.
.Indent	If .Hang is set to 1, the width of the left indent in points or a text measurement.
.Remove	Removes existing bullets or numbers.

See also

Bullets and Numbering Statements and Functions

FormatBulletDefault

FormatBulletsAndNumbering

FormatNumberDefault

ToolsBulletListDefault

ToolsNumberListDefault

ToolsNumberListDefault

Adds numbers and tab characters to the selected paragraphs and formats the paragraphs with a hanging indent.

Note

The ToolsNumberListDefault statement corresponds to the Numbered List button on the Toolbar in Word version 2.x. In Word version 6.0, the Numbering button is on the Formatting toolbar and its corresponding WordBasic statement is FormatNumberDefault.

See also

Bullets and Numbering Statements and Functions

FormatBulletDefault

FormatBulletsAndNumbering

FormatNumberDefault

ToolsBulletListDefault

ToolsBulletsNumbers

What's New in WordBasic

New Macro-Editing and WordBasic Capabilities

This section describes improvements to the macro-editing environment and WordBasic capabilities that were not available in earlier versions of Word.

New Macro Toolbar Buttons

The Macro Text Style

Global Templates

The Organizer Dialog Box

New Custom Dialog Box Capabilities

Miscellaneous Improvements

New WordBasic Statements and Functions

This section lists new or modified WordBasic statements and functions, sorted by category. Note that statements and functions that correspond to new commands, toolbar buttons, and other new features of Word version 6.0 are not listed.

Application Control Statements and Functions

Date and Time Functions

Disk Access Statements and Functions

Environment Statements and Functions

Menu Customization Statements and Functions

Selection Statements and Functions

String Functions

Window Control Statements and Functions

Miscellaneous Statements and Functions

New Macro Toolbar Buttons

The Macro toolbar now includes graphical buttons and a box you can use to select any open macro to run. The following new toolbar buttons correspond to features not accessible from the Macro toolbar in earlier versions of Word.

Click	To
ewc shareres, T3EWCLASS, dllres:wordres. dll:TBAR8:0:0: 16:0	Display the Record Macro dialog box.
ewc shareres, T3EWCLASS, dllres:wordres. dll:TBAR8:16: 0:16:0	Record the next command you perform.
ewc shareres, T3EWCLASS, dllres:wordres. dll:TBAR8:64: 0:16:0	Now allows you to step through subroutines and functions in other open macros.
ewc shareres, T3EWCLASS, dllres:wordres. dll:TBAR8:128: 0:16:0	Add or remove "REM " from the beginning of selected lines.
ewc shareres, T3EWCLASS, dllres:wordres. dll:TBAR8:96: 0:16:0	Display the Macro dialog box (Tools menu).
ewc shareres, T3EWCLASS, dllres:wordres. dll:TBAR8:112: 0:16:0	Open the Dialog Editor.

The Macro Text Style

You can use the new Macro Text built-in style to change the default style of text in a macro-editing window. For example, you can change the font or the tab stop settings.

Global Templates

Using the Templates command (File menu), you can make any template a global template. The macros in a global template are available in any document window, just like macros stored in the Normal template. This means that you can access the macros stored in a template without having to attach the template to a document or base a document on it.

The Organizer Dialog Box

You can use the new Organizer dialog box to manage your macros. You can select any number of macros in a template and copy or move them to another template; you can rename macros or delete them. You display the Organizer dialog box by choosing the Organizer button in the Macro dialog box (Tools menu).

New Custom Dialog Box Capabilities

WordBasic supports four new controls for custom dialog boxes:

- Drop-down list boxes are supported with the DropListBox statement.
- Multiple-line text boxes are supported with a new argument for the TextBox statement.
- Graphics are supported with the Picture statement.
- File preview boxes are supported with the FilePreview statement.

In addition, you can now create dialog boxes that change dynamically. For example, you can create a dialog box that updates the contents of a list box based on options the user selects elsewhere in the dialog box. Also, there is no longer any limit to the number of controls a custom dialog box can contain. For more information about dynamic dialog boxes, see Creating Dynamic Dialog Boxes and Dialog Function Syntax.

Miscellaneous Improvements

The following improvements have been made to WordBasic:

- The ability to turn off screen updates. You can use the ScreenUpdating statement to control whether changes are displayed on your monitor while a macro is running. You can increase the speed of some macros by preventing screen updates.
- New date and time functions. WordBasic now includes a set of "serial number" date and time functions compatible with Visual Basic version 3.0. In addition, the Date\$() and Time\$() functions have been modified to accept serial numbers for dates and times.
- Improved array handling. You can now pass array variables to subroutines and user-defined functions. You can use the SortArray statement to sort arrays. For more information about arrays, see SortArray and Chapter 4, "Advanced WordBasic," in the Microsoft Word Developer's Kit.
- Server support for object linking and embedding (OLE) Automation. Applications that support OLE Automation, such as Microsoft Excel version 5.0, can use OLE Automation to access Word. For more information about OLE Automation, see Chapter 8, "Communicating with Other Applications," in the Microsoft Word Developer's Kit.
- Private settings files. Using SetPrivateProfileString and GetPrivateProfileString\$(), you can create private settings files to store variables and values. For more information about private settings files, see SetPrivateProfileString, GetPrivateProfileString\$(), and Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.
- Document variables. Using SetDocumentVar and GetDocumentVar\$(), you can store and retrieve variables in the active document. For more information about document variables, see SetDocumentVar, GetDocumentVar\$(), and Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.
- Form-field macros. You can attach macros to form fields so that macros are triggered either when a form field is activated (an "on-entry" macro) or when it is no longer active (an "on-exit" macro). For more information about form-field macros, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.
- Larger variables. String variables can now hold as many as 64K characters; most string functions now accept 64K strings. A numeric variable can be as large as 1.79E+308.
- The Stop statement, used to interrupt a macro, now includes an argument to suspend the macro without displaying an error message. Usually, when you are debugging, the error message is unnecessary. For more information on debugging, see Chapter 6, "Debugging," in the Microsoft Word Developer's Kit.

Application Control Statements and Functions

<u>AppClose</u>	Closes the specified application
<u>AppCount()</u>	Returns the number of open applications (including hidden applications that do not appear in the Task List)
<u>AppGetNames</u>	Fills an array with the names of open application windows
<u>AppHide</u>	Hides the specified application and removes its window name from the Task List
<u>AppIsRunning()</u>	Returns -1 if the specified application is running or 0 (zero) if it is not
<u>AppSendMessage</u>	Sends a Windows message and its associated parameters to the specified application
<u>AppShow</u>	Makes visible and activates an application previously hidden with AppHide and restores the application window name to the Task List

Date and Time Functions

<u>Date\$()</u>	Now takes a serial number as an optional argument
<u>DateSerial()</u>	Returns the serial number of a date specified in the format Year, Month, Day
<u>DateValue()</u>	Returns the serial number of a date specified as a string
<u>Day()</u>	Returns the day of the month corresponding to the specified serial number
<u>Days360()</u>	Returns the number of days between two dates based on a 360-day year (twelve 30-day months)
<u>Hour()</u>	Returns the hours component of the specified serial number
<u>Minute()</u>	Returns the minutes component of the specified serial number
<u>Month()</u>	Returns the month component of the specified serial number
<u>Now()</u>	Returns a serial number that represents the current date and time
<u>Second()</u>	Returns the seconds component of the specified serial number
<u>Time\$()</u>	Now takes a serial number as an optional argument
<u>TimeSerial()</u>	Returns the serial number of a time specified in the format Hour, Minute, Second
<u>TimeValue()</u>	Returns the serial number of a time specified as a string
<u>Today()</u>	Returns a serial number that represents the current date
<u>Weekday()</u>	Returns a number corresponding to the day of the week on which the date represented by the specified serial number falls
<u>Year()</u>	Returns the year component of the specified serial number

Disk Access Statements and Functions

<u>CountDirectories()</u>	Returns the number of subdirectories contained within the specified directory
<u>GetAttr()</u>	Returns a number corresponding to the file attributes set for the specified file
<u>GetDirectory\$()</u>	Returns the name of a subdirectory within the specified directory
<u>SetAttr</u>	Sets file attributes for the specified file

Environment Statements and Functions

<u>Environ\$()</u>	Returns a string associated with an MS-DOS environment variable
<u>GetSystemInfo</u>	Fills a string array with each available piece of information about the environment in which Word is running
<u>GetSystemInfo\$()</u>	Returns one piece of information about the environment in which Word is running

Menu Customization Statements and Functions

<u>CountMenuItems()</u>	Now returns the number of all menu items on the specified menu, not just those that differ from the defaults
<u>CountMenus()</u>	Returns the number of menus of the specified type
<u>MenuItemMacro\$()</u> , <u>MenuItemText\$()</u>	Now return information about any menu item, not just those that differ from the defaults. Note that these functions were previously MenuMacro\$() and MenuText\$().
<u>MenuText\$()</u>	Now returns the name of a shortcut menu or a menu on the menu bar instead of the text of a menu item.
<u>ToolsCustomizeMenuBar</u>	Adds, removes, or renames menus on the menu bar.

Selection Statements and Functions

<u>GetSelEndPos</u>	Returns the character position of the end of the selection relative to the start of the document
<u>GetSelStartPos</u>	Returns the character position of the start of the selection relative to the start of the document
<u>GetText\$()</u>	Returns the text (unformatted) between and including two specified character positions
<u>SelectCurAlignment,</u> <u>SelectCurColor,</u> <u>SelectCurFont,</u> <u>SelectCurIndent,</u> <u>SelectCurSpacing,</u> <u>SelectCurTabs</u>	Extend the selection forward until text with different settings for alignment, color, font, indents, spacing, or tab stops is encountered
<u>SelectCurSentence</u>	Selects the entire sentence containing the insertion point
<u>SelectCurWord</u>	Selects the entire word containing the insertion point
<u>SetSelRange</u>	Selects characters between two specified character positions relative to the start of the document

String Functions

<u>CleanString\$()</u>	Changes nonprinting characters and special Word characters to spaces (ANSI character code 32)
<u>DOSToWin\$()</u> , <u>WinToDOS\$()</u>	Translate a string from the original equipment manufacturer (OEM) character set to the Windows character set, and vice versa
<u>LTrim\$()</u> , <u>RTrim\$()</u>	Remove leading and trailing spaces, respectively, from a specified string

Window Control Statements and Functions

AppMaximize,
AppMinimize,
AppMove,
AppRestore,
AppSize

Now take an optional argument for specifying any open application window. AppMove and AppSize now use points as the unit of measure. AppMaximize, AppMinimize, and AppRestore have corresponding functions.

AppWindowHeight,
AppWindowWidth,
DocWindowHeight,
DocWindowWidth

Set the height of a window (in points) without affecting the width, and vice versa. All these statements have corresponding functions.

AppWindowPosLeft,
AppWindowPosTop,
DocWindowPosLeft,
DocWindowPosTop

Set the horizontal position of a window (in points) without affecting the vertical position, and vice versa. All these statements have corresponding functions.

Miscellaneous Statements and Functions

<u>CountDocumentVars()</u>	Manage document variables.
<u>GetDocumentVar\$()</u>	
<u>GetDocumentVarName\$()</u>	
<u>SetDocumentVar</u>	
<u>FileNameInfo\$()</u>	Returns all or part of the path and filename of the specified file.
<u>GetPrivateProfileString\$()</u>	Store values in private settings files; retrieve values from private settings files.
<u>SetPrivateProfileString</u>	
<u>IsTemplateDirty()</u>	Returns a value indicating whether the active template has changed since it was last saved. Note that <u>IsDirty()</u> has changed to <u>IsDocumentDirty()</u> .
<u>PathFromMacPath\$()</u>	Converts a Macintosh path and filename to a valid path and filename for the current operating system.
<u>SaveTemplate</u>	Saves changes to the active template or the global template.
<u>ScreenUpdating</u>	Controls whether changes are displayed on your monitor while a macro is running.
<u>SelectionFileName\$()</u>	Returns the full path and filename of the active document if it has been saved. If the document has not been saved, or if the active window is a macro-editing window, returns the current path followed by a backslash (\).
<u>SetTemplateDirty</u>	Controls whether Word recognizes a template as changed since the last time the template was saved. Note that <u>SetDirty</u> has changed to <u>SetDocumentDirty</u> .
<u>SortArray</u>	Sorts the elements in a specified numeric or string array alphanumerically. This statement is especially useful for sorting arrays that fill list boxes in a user-defined dialog box.
<u>Stop</u>	Now includes an argument to suspend the macro without displaying an error message.
<u>WaitCursor</u>	Changes the mouse pointer from the current pointer to an hourglass, or vice versa.

WordBasic Statements and Functions by Category

WordBasic keywords are listed here by category. Refer to this section when you know what you want to do but not which commands you need to accomplish the task, or when you want to learn about related statements and functions. Keywords appear alphabetically in each list; some keywords appear in more than one category.

Application Control

AutoCorrect

AutoText

Basic File Input/Output

Bookmarks

Borders and Frames

Branching and Control

Bullets and Numbering

Character Formatting

Customization

Date and Time

Definitions and Declarations

Dialog Box Definition and Control

Disk Access and Management

Documents, Templates, and Add-ins

Drawing

Dynamic Data Exchange (DDE)

Editing

Environment

Fields

Finding and Replacing

Footnotes, Endnotes, and Annotations

Forms

Help

Macros

Mail Merge

Moving the Insertion Point and Selecting

Object Linking and Embedding

Outlining and Master Documents

Paragraph Formatting

Proofing

Section and Document Formatting

Strings and Numbers

Style Formatting

Tables

Tools

View

Windows

Application Control

AppActivate

AppClose

AppCount()

AppGetNames, AppGetNames()

AppHide

AppInfo\$()

AppIsRunning()

AppMaximize, AppMaximize()

AppMinimize, AppMinimize()

AppMove

AppRestore, AppRestore()

AppSendMessage

AppShow

AppSize

AppWindowHeight, AppWindowHeight()

AppWindowPosLeft, AppWindowPosLeft()

AppWindowPosTop, AppWindowPosTop()

AppWindowWidth, AppWindowWidth()

ControlRun

DDEExecute

DDEInitiate()

DDEPoke

DDERequest\$()

DDETerminate

DDETerminateAll

DialogEditor

ExitWindows

FileExit

GetSystemInfo, GetSystemInfo\$()

MicrosoftAccess

MicrosoftExcel

MicrosoftFoxPro

MicrosoftMail

MicrosoftPowerPoint

MicrosoftProject

MicrosoftPublisher

MicrosoftSchedule

MicrosoftSystemInfo

RunPrintManager

SendKeys

Shell

AutoCorrect

GetAutoCorrect\$()

ToolsAutoCorrect

ToolsAutoCorrectDays, ToolsAutoCorrectDays()

ToolsAutoCorrectInitialCaps, ToolsAutoCorrectInitialCaps()

ToolsAutoCorrectReplaceText, ToolsAutoCorrectReplaceText()

ToolsAutoCorrectSentenceCaps, ToolsAutoCorrectSentenceCaps()

ToolsAutoCorrectSmartQuotes, ToolsAutoCorrectSmartQuotes()

AutoText

AutoText

AutoTextName\$()

CountAutoTextEntries()

EditAutoText

GetAutoText\$()

InsertAutoText

Organizer

SetAutoText

Basic File Input/Output

Close

Eof()

Input

Input\$()

Line Input

Lof()

Open

Print

Read

Seek, Seek()

Write

Bookmarks

BookmarkName\$()

CmpBookmarks()

CopyBookmark

CountBookmarks()

EditBookmark

EmptyBookmark()

ExistingBookmark()

GetBookmark\$()

SetEndOfBookmark

SetStartOfBookmark

Borders and Frames

BorderBottom, BorderBottom()

BorderInside, BorderInside()

BorderLeft, BorderLeft()

BorderLineStyle, BorderLineStyle()

BorderNone, BorderNone()

BorderOutside, BorderOutside()

BorderRight, BorderRight()

BorderTop, BorderTop()

FormatBordersAndShading

FormatDefineStyleBorders

FormatDefineStyleFrame

FormatFrame

InsertFrame

RemoveFrames

ShadingPattern, ShadingPattern()

ViewBorderToolbar

Branching and Control

Call

For...Next

Function...End Function

Goto

If...Then...Else

On Error

OnTime

Select Case

Stop

Sub...End Sub

While...Wend

Bullets and Numbering

DemoteList

FormatBullet

FormatBulletDefault, FormatBulletDefault()

FormatBulletsAndNumbering

FormatDefineStyleNumbers

FormatMultilevel

FormatNumber

FormatNumberDefault, FormatNumberDefault()

PromoteList

RemoveBulletsNumbers

SkipNumbering, SkipNumbering()

ToolsBulletListDefault

ToolsBulletsNumbers

ToolsNumberListDefault

Character Formatting

AllCaps, AllCaps()

Bold, Bold()

CharColor, CharColor()

CopyFormat

CountFonts()

CountLanguages()

DottedUnderline, DottedUnderline()

DoubleUnderline, DoubleUnderline()

Font, Font\$()

FontSize, FontSize()

FontSizeSelect

FontSubstitution

FormatAddrFonts

FormatChangeCase

FormatDefineStyleFont

FormatDefineStyleLang

FormatFont

FormatRetAddrFonts

GrowFont

GrowFontOnePoint

Hidden, Hidden()

Italic, Italic()

Language, Language\$()

NormalFontPosition

NormalFontSpacing

PasteFormat

ResetChar, ResetChar()

ShrinkFont

ShrinkFontOnePoint

SmallCaps, SmallCaps()

Strikethrough, Strikethrough()

Subscript, Subscript()

Superscript, Superscript()

SymbolFont

ToolsLanguage

Underline, Underline()

WordUnderline, WordUnderline()

Customization

AddButton

ChooseButtonImage

CopyButtonImage

CountKeys()

CountMenuItems()

CountMenus()

CountToolbarButtons()

CountToolbars()

DeleteButton

EditButtonImage

KeyCode()

KeyMacro\$()

MenuItemMacro\$()

MenuItemText\$()

MenuMode

MenuText\$()

MoveButton

MoveToolbar

NewToolbar

PasteButtonImage

RenameMenu

ResetButtonImage

SizeToolbar

ToolbarButtonMacro\$()

ToolbarName\$()

ToolbarState()

ToolsCustomize

ToolsCustomizeKeyboard

ToolsCustomizeMenuBar

ToolsCustomizeMenus

Date and Time

Date\$()

DateSerial()

DateValue()

Day()

Days360()

Hour()

InsertDateField

InsertDateTime

InsertTimeField

Minute()

Month()

Now()

OnTime

Second()

Time\$()

TimeSerial()

TimeValue()

Today()

ToolsRevisionDate()

ToolsRevisionDate\$()

Weekday()

Year()

Definitions and Declarations

Declare

Dim

Let

Redim

Dialog Box Definition and Control

Begin Dialog...End Dialog

CancelButton

CheckBox

ComboBox

Dialog, Dialog()

DialogEditor

DlgControlId()

DlgEnable, DlgEnable()

DlgFilePreview, DlgFilePreview\$()

DlgFocus, DlgFocus\$()

DlgListBoxArray, DlgListBoxArray()

DlgSetPicture

DlgText, DlgText\$()

DlgUpdateFilePreview

DlgValue, DlgValue()

DlgVisible, DlgVisible()

DropListBox

FilePreview

GetCurValues

GroupBox

InputBox\$()

ListBox

MsgBox, MsgBox()

OKButton

OptionButton

OptionGroup

Picture

PushButton

Text

TextBox

Disk Access and Management

ChDefaultDir

ChDir

Connect

CopyFile

CountDirectories()

DefaultDir\$()

Files\$()

GetAttr()

GetDirectory\$()

Kill

MkDir

Name

RmDir

SetAttr

Documents, Templates, and Add-ins

AddAddIn, AddAddIn()

AddInState, AddInState()

ClearAddIns

Converter\$()

ConverterLookup()

CopyFile

CountAddIns()

CountDocumentVars()

CountFiles()

CountFoundFiles()

DeleteAddIn

DisableInput

DocClose

DocumentStatistics

FileClose

FileCloseAll

FileConfirmConversions, FileConfirmConversions()

FileFind

FileList

FileName\$()

FileNameFromWindow\$()

FileNameInfo\$()

FileNew

FileNewDefault

FileNumber

FileOpen

FilePageSetup

FilePrint

FilePrintDefault

FilePrintPreview, FilePrintPreview()

FilePrintPreviewFullScreen

FilePrintPreviewPages, FilePrintPreviewPages()

FilePrintSetup

FileRoutingSlip

Files\$()

FileSave

FileSaveAll

FileSaveAs

FileSendMail

FileSummaryInfo

FileTemplates

FoundFileName\$()

GetAddInID()

GetAddInName\$()

GetAttr()

GetDocumentVar\$()

GetDocumentVarName\$()

InsertFile

Kill

LockDocument, LockDocument()

MacroFileName\$()

Name

Organizer

PathFromMacPath\$()

SaveTemplate

SelectionFileName\$()

SetAttr

SetDocumentVar, SetDocumentVar()

ToolsOptionsFileLocations

ToolsOptionsPrint

Drawing

DrawAlign

DrawArc

DrawBringForward

DrawBringInFrontOfText

DrawBringToFront

DrawCallout

DrawClearRange

DrawCount()

DrawCountPolyPoints()

DrawDisassemblePicture

DrawEllipse

DrawExtendSelect

DrawFlipHorizontal

DrawFlipVertical

DrawFreeformPolygon

DrawGetCalloutTextbox

DrawGetPolyPoints

DrawGetType()

DrawGroup

DrawInsertWordPicture

DrawLine

DrawNudgeDown

DrawNudgeDownPixel

DrawNudgeLeft

DrawNudgeLeftPixel

DrawNudgeRight

DrawNudgeRightPixel

DrawNudgeUp

DrawNudgeUpPixel

DrawRectangle

DrawResetWordPicture

DrawReshape

DrawRotateLeft

DrawRotateRight

DrawRoundRectangle

DrawSelect, DrawSelect()

DrawSelectNext

DrawSelectPrevious

DrawSendBackward

DrawSendBehindText

DrawSendToBack

DrawSetCalloutTextbox

DrawSetInsertToAnchor

DrawSetInsertToTextbox

DrawSetPolyPoints

DrawSetRange, DrawSetRange()

DrawSnapToGrid

DrawTextBox

DrawUngroup
DrawUnselect
FormatCallout
FormatDrawingObject
FormatPicture
InsertDrawing
SelectDrawingObjects
ToggleScribbleMode
ViewDrawingToolbar

Dynamic Data Exchange (DDE)

DDEExecute

DDEInitiate()

DDEPoke

DDERequest\$()

DDETerminate

DDETerminateAll

SendKeys

Editing

AutoMarkIndexEntries

Cancel

ChangeCase, ChangeCase()

CopyText

DeleteBackWord

DeleteWord

EditClear

EditCopy

EditCut

EditFind

EditGoTo

EditLinks

EditObject

EditPaste

EditPasteSpecial

EditPicture

EditRedo

EditRepeat

EditReplace

EditTOACategory

EditUndo

ExtendMode()

Insert

InsertAddCaption

InsertAutoCaption

InsertBreak

InsertCaption

InsertCaptionNumbering

InsertColumnBreak

InsertCrossReference

InsertIndex

InsertPageBreak

InsertPageNumbers

InsertSpike

InsertSymbol

InsertTableOfAuthorities

InsertTableOfContents

InsertTableOfFigures

MarkCitation

MarkIndexEntry

MarkTableOfContentsEntry

MoveText

OK

Overtyping, Overtyping()

Spike

ToolsOptionsEdit

Environment

AppInfo\$()

Beep

CommandValid()

DOSToWin\$()

Environ\$()

Err

Error

GetPrivateProfileString\$()

GetProfileString\$()

GetSystemInfo, GetSystemInfo\$()

IsDocumentDirty()

IsExecuteOnly()

IsMacro()

IsTemplateDirty()

LockDocument, LockDocument()

MacroFileName\$()

MicrosoftSystemInfo

ScreenRefresh

ScreenUpdating, ScreenUpdating()

SelInfo()

SelType, SelType()

SetDocumentDirty

SetPrivateProfileString, SetPrivateProfileString()

SetProfileString

SetTemplateDirty

ViewMenus()

WaitCursor

WinToDOS\$()

Fields

CheckBoxFormField

CountMergeFields()

DoFieldClick

DropDownFormField

EnableFormField

FormFieldOptions

GetFieldData\$()

GetMergeField\$()

InsertDateField

InsertDateTime

InsertField

InsertFieldChars

InsertFormField

InsertMergeField

InsertPageField

InsertTimeField

LockFields

MergeFieldName\$()

NextField, NextField()

PrevField, PrevField()

PutFieldData

TextFormField

ToggleFieldDisplay

ToolsManageFields

UnlinkFields

UnlockFields

UpdateFields

UpdateSource

ViewFieldCodes, ViewFieldCodes()

Finding and Replacing

EditFind

EditFindClearFormatting

EditFindFont

EditFindFound()

EditFindLang

EditFindPara

EditFindStyle

EditReplace

EditReplaceClearFormatting

EditReplaceFont

EditReplaceLang

EditReplacePara

EditReplaceStyle

RepeatFind

Footnotes, Endnotes, and Annotations

AnnotationRefFromSel\$()

EditConvertAllEndnotes

EditConvertAllFootnotes

EditConvertNotes

EditSwapAllNotes

GoToAnnotationScope

InsertAnnotation

InsertFootnote

NoteOptions

ResetNoteSepOrNotice

ShowAnnotationBy

ViewAnnotations, ViewAnnotations()

ViewEndnoteArea, ViewEndnoteArea()

ViewEndnoteContNotice

ViewEndnoteContSeparator

ViewEndnoteSeparator

ViewFootnoteArea, ViewFootnoteArea()

ViewFootnoteContNotice

ViewFootnoteContSeparator

ViewFootnotes, ViewFootnotes()

ViewFootnoteSeparator

Forms

AddDropDownItem

CheckBoxFormField

ClearFormField

DropDownFormField

EnableFormField

FormFieldOptions

FormShading, FormShading()

GetFormResult(), GetFormResult\$()

InsertFormField

RemoveAllDropDownItems

RemoveDropDownItem

SetFormResult

TextFormField

ToolsProtectDocument

ToolsProtectSection

ToolsUnprotectDocument

[Help](#)

[Help](#)

[HelpAbout](#)

[HelpActiveWindow](#)

[HelpContents](#)

[HelpExamplesAndDemos](#)

[HelpIndex](#)

[HelpKeyboard](#)

[HelpPSSHelp](#)

[HelpQuickPreview](#)

[HelpSearch](#)

[HelpTipOfTheDay](#)

[HelpTool](#)

[HelpUsingHelp](#)

[HelpWordPerfectHelp](#)

[HelpWordPerfectHelpOptions](#)

Macros

CommandValid()

CountMacros()

DisableAutoMacros

IsExecuteOnly()

IsMacro()

KeyMacro\$()

MacroCopy

MacroDesc\$()

MacroFileName\$()

MacroName\$()

MacroNameFromWindow\$()

MenuItemMacro\$()

OnTime

Organizer

PauseRecorder

REM

ShowVars

ToolbarButtonMacro\$()

ToolsMacro

Mail Merge

CountMergeFields()

GetMergeField\$()

InsertMergeField

MailMerge

MailMergeAskToConvertChevrons, MailMergeAskToConvertChevrons()

MailMergeCheck

MailMergeConvertChevrons, MailMergeConvertChevrons()

MailMergeCreateDataSource

MailMergeCreateHeaderSource

MailMergeDataForm

MailMergeDataSource\$()

MailMergeEditDataSource

MailMergeEditHeaderSource

MailMergeEditMainDocument

MailMergeFindRecord

MailMergeFirstRecord

MailMergeFoundRecord()

MailMergeGotoRecord, MailMergeGotoRecord()

MailMergeHelper

MailMergeInsertAsk

MailMergeInsertFillIn

MailMergeInsertIf

MailMergeInsertMergeRec

MailMergeInsertMergeSeq

MailMergeInsertNext

MailMergeInsertNextIf

MailMergeInsertSet

MailMergeInsertSkipIf

MailMergeLastRecord

MailMergeMainDocumentType, MailMergeMainDocumentType()

MailMergeNextRecord

MailMergeOpenDataSource

MailMergeOpenHeaderSource

MailMergePrevRecord

MailMergeQueryOptions

MailMergeReset

MailMergeState()

MailMergeToDoc

MailMergeToPrinter

MailMergeViewData, MailMergeViewData()

MergeFieldName\$()

ToolsAddRecordDefault

ToolsRemoveRecordDefault

Moving the Insertion Point and Selecting

AtEndOfDocument()

AtStartOfDocument()

Cancel

CharLeft, CharLeft()

CharRight, CharRight()

ColumnSelect

EditSelectAll

EndOfColumn, EndOfColumn()

EndOfDocument, EndOfDocument()

EndOfLine, EndOfLine()

EndOfRow, EndOfRow()

EndOfWindow, EndOfWindow()

ExtendMode()

ExtendSelection

GetSelEndPos()

GetSelStartPos()

GetText\$()

GoBack

GoToAnnotationScope

GoToHeaderFooter

GoToNextItem

GoToPreviousItem

HLine

HPage

HScroll, HScroll()

Insert

LineDown, LineDown()

LineUp, LineUp()

NextCell, NextCell()

NextField, NextField()

NextObject

NextPage, NextPage()

NextWindow

OtherPane

PageDown, PageDown()

PageUp, PageUp()

ParaDown, ParaDown()

ParaUp, ParaUp()

PrevCell, PrevCell()

PrevField, PrevField()

PrevObject

PrevPage, PrevPage()

PrevWindow

SelectCurAlignment

SelectCurColor

SelectCurFont

SelectCurIndent

SelectCurSentence

SelectCurSpacing
SelectCurTabs
SelectCurWord
SelType, SelType()
SentLeft, SentLeft()
SentRight, SentRight()
SetSelRange
ShrinkSelection
StartOfColumn, StartOfColumn()
StartOfDocument, StartOfDocument()
StartOfLine, StartOfLine()
StartOfRow, StartOfRow()
StartOfWindow, StartOfWindow()
TableSelectColumn
TableSelectRow
TableSelectTable
VLine
VPage
VScroll, VScroll()
WordLeft, WordLeft()
WordRight, WordRight()

Object Linking and Embedding

ActivateObject

ConvertObject

EditLinks

EditObject

EditPasteSpecial

EditPicture

FileClosePicture

InsertChart

InsertDatabase

InsertEquation

InsertExcelTable

InsertObject

InsertPicture

InsertSound

InsertWordArt

Outlining and Master Documents

CreateSubdocument

DemoteToBodyText

InsertSubdocument

MergeSubdocument

OpenSubdocument

OutlineCollapse

OutlineDemote

OutlineExpand

OutlineLevel()

OutlineMoveDown

OutlineMoveUp

OutlinePromote

OutlineShowFirstLine, OutlineShowFirstLine()

OutlineShowFormat

RemoveSubdocument

ShowAllHeadings

ShowHeadingNumber

SplitSubdocument

ViewMasterDocument, ViewMasterDocument()

ViewOutline, ViewOutline()

ViewToggleMasterDocument

Paragraph Formatting

CenterPara, CenterPara()

CloseUpPara

CopyFormat

FormatDefineStylePara

FormatDefineStyleTabs

FormatDropCap

FormatParagraph

FormatTabs

HangingIndent

Indent

InsertPara

JustifyPara, JustifyPara()

LeftPara, LeftPara()

NextTab()

OpenUpPara

ParaKeepLinesTogether, ParaKeepLinesTogether()

ParaKeepWithNext, ParaKeepWithNext()

ParaPageBreakBefore, ParaPageBreakBefore()

ParaWidowOrphanControl, ParaWidowOrphanControl()

PasteFormat

PrevTab()

ResetPara, ResetPara()

RightPara, RightPara()

SpacePara1, SpacePara1()

SpacePara15, SpacePara15()

SpacePara2, SpacePara2()

TabLeader\$()

TabType()

UnHang

UnIndent

Proofing

CountToolsGrammarStatistics()

ToolsGetSpelling, ToolsGetSpelling()

ToolsGetSynonyms, ToolsGetSynonyms()

ToolsGrammar

ToolsGrammarStatisticsArray

ToolsHyphenation

ToolsHyphenationManual

ToolsOptionsGrammar

ToolsOptionsSpelling

ToolsSpelling

ToolsSpellSelection

ToolsThesaurus

Section and Document Formatting

CloseViewHeaderFooter

FormatAutoFormat

FormatColumns

FormatHeaderFooterLink

FormatHeadingNumber

FormatHeadingNumbering

FormatPageNumber

FormatSectionLayout

GoToHeaderFooter

InsertSectionBreak

ShowNextHeaderFooter

ShowPrevHeaderFooter

ToggleHeaderFooterLink

ToggleMainTextLayer

TogglePortrait

ToolsOptionsAutoFormat

ViewFooter, ViewFooter()

ViewHeader, ViewHeader()

Strings and Numbers

Abs()

Asc()

Chr\$()

CleanString\$()

InStr()

Int()

LCase\$()

Left\$()

Len()

LTrim\$()

Mid\$()

Right\$()

Rnd()

RTrim\$()

Selection\$()

Sgn()

SortArray

Str\$()

String\$()

UCase\$()

Val()

Style Formatting

CountStyles()

FormatDefineStyleBorders

FormatDefineStyleFont

FormatDefineStyleFrame

FormatDefineStyleLang

FormatDefineStyleNumbers

FormatDefineStylePara

FormatDefineStyleTabs

FormatStyle

FormatStyleGallery

NormalStyle

Organizer

Style

StyleDesc\$()

StyleName\$()

Tables

FieldSeparator\$, FieldSeparator\$()

InsertExcelTable

NextCell, NextCell()

PrevCell, PrevCell()

TableAutoFormat

TableAutoSum

TableColumnWidth

TableDeleteCells

TableDeleteColumn

TableDeleteRow

TableFormula

TableGridlines, TableGridlines()

TableHeadings, TableHeadings()

TableInsertCells

TableInsertColumn

TableInsertRow

TableInsertTable

TableMergeCells

TableRowHeight

TableSelectColumn

TableSelectRow

TableSelectTable

TableSort

TableSortAToZ

TableSortZToA

TableSplit

TableSplitCells

TableToText

TableUpdateAutoFormat

TextToTable

Tools

ToolsAdvancedSettings

ToolsCalculate, ToolsCalculate()

ToolsCompareVersions

ToolsCreateEnvelope

ToolsCreateLabels

ToolsCustomize

ToolsHyphenation

ToolsHyphenationManual

ToolsLanguage

ToolsMergeRevisions

ToolsOptions

ToolsOptionsAutoFormat

ToolsOptionsCompatibility

ToolsOptionsEdit

ToolsOptionsFileLocations

ToolsOptionsGeneral

ToolsOptionsPrint

ToolsOptionsRevisions

ToolsOptionsSave

ToolsOptionsUserInfo

ToolsOptionsView

ToolsProtectDocument

ToolsProtectSection

ToolsRepaginate

ToolsReviewRevisions

ToolsRevisionAuthor\$()

ToolsRevisionDate()

ToolsRevisionDate\$()

ToolsRevisions

ToolsRevisionType()

ToolsShrinkToFit

ToolsUnprotectDocument

ToolsWordCount

View

ClosePreview

CloseViewHeaderFooter

FilePrintPreview, FilePrintPreview()

FilePrintPreviewFullScreen

FilePrintPreviewPages, FilePrintPreviewPages()

Magnifier, Magnifier()

ShowAll, ShowAll()

ShowNextHeaderFooter

ShowPrevHeaderFooter

ToggleFull

TogglePortrait

ToolsOptionsView

ViewAnnotations, ViewAnnotations()

ViewBorderToolbar

ViewDraft, ViewDraft()

ViewDrawingToolbar

ViewEndnoteArea, ViewEndnoteArea()

ViewEndnoteContNotice

ViewEndnoteContSeparator

ViewEndnoteSeparator

ViewFieldCodes, ViewFieldCodes()

ViewFooter, ViewFooter()

ViewFootnoteArea, ViewFootnoteArea()

ViewFootnoteContNotice

ViewFootnoteContSeparator

ViewFootnotes, ViewFootnotes()

ViewFootnoteSeparator

ViewHeader, ViewHeader()

ViewMasterDocument, ViewMasterDocument()

ViewMenus()

ViewNormal, ViewNormal()

ViewOutline, ViewOutline()

ViewPage, ViewPage()

ViewRibbon, ViewRibbon()

ViewRuler, ViewRuler()

ViewStatusBar, ViewStatusBar()

ViewToggleMasterDocument

ViewToolbars

ViewZoom

ViewZoom100

ViewZoom200

ViewZoom75

ViewZoomPageWidth

ViewZoomWholePage

Windows

Activate

AppActivate

AppClose

AppCount()

AppGetNames, AppGetNames()

AppHide

AppMaximize, AppMaximize()

AppMinimize, AppMinimize()

AppMove

AppRestore, AppRestore()

AppShow

AppSize

AppWindowHeight, AppWindowHeight()

AppWindowPosLeft, AppWindowPosLeft()

AppWindowPosTop, AppWindowPosTop()

AppWindowWidth, AppWindowWidth()

ClosePane

CountWindows()

DocClose

DocMaximize, DocMaximize()

DocMinimize, DocMinimize()

DocMove

DocRestore

DocSize

DocSplit, DocSplit()

DocWindowHeight, DocWindowHeight()

DocWindowPosLeft, DocWindowPosLeft()

DocWindowPosTop, DocWindowPosTop()

DocWindowWidth, DocWindowWidth()

ExitWindows

FileNameFromWindow\$()

HelpActiveWindow

IsMacro()

NextWindow

OtherPane

PrevWindow

Window()

WindowArrangeAll

WindowList

WindowName\$()

WindowNewWindow

WindowNumber

WindowPane()

Error Messages

When you run a macro and an error occurs, you can get more information by pressing F1 or choosing the Help button in the error message box. The following lists, the first for WordBasic error messages and the second for Word error messages, includes numbers you can use when trapping errors. For more information on error trapping, see On Error statement.

WordBasic Error Messages

Error #	Message
5	<u>Illegal function call</u>
6	<u>Overflow</u>
7	<u>Out of memory</u>
9	<u>Subscript out of range</u>
11	<u>Division by zero</u>
14	<u>Out of string space</u>
22	<u>Invalid array dimension</u>
24	<u>Bad parameter</u>
25	<u>Out of memory (stack space)</u>
26	<u>Dialog needs End Dialog or a push button</u>
28	<u>Directory already exists</u>
39	<u>CASE ELSE expected</u>
51	<u>Internal error</u>
52	<u>Bad file name or number</u>
53	<u>File not found</u>
54	<u>Bad file mode</u>
55	<u>File already open</u>
57	<u>Device I/O error</u>
62	<u>Input past end of file</u>
64	<u>Bad file name</u>
67	<u>Too many files</u>
74	<u>Rename across disks</u>
75	<u>Path/File access error</u>
76	<u>Path not found</u>
100	<u>Syntax error</u>
101	<u>Comma missing</u>
102	<u>Command failed</u>
103	<u>Dialog record variable expected</u>
104	<u>ELSE without IF</u>
105	<u>END IF without IF</u>
109	<u>INPUT missing</u>
111	<u>Expression too complex</u>
112	<u>Identifier expected</u>
113	<u>Duplicate label</u>
114	<u>Label not found</u>
115	<u>Right parenthesis missing</u>
116	<u>Argument-count mismatch</u>
117	<u>Missing NEXT or WEND</u>
118	<u>Nested SUB or FUNCTION definitions</u>
119	<u>NEXT without FOR</u>
120	<u>Array already dimensioned</u>
122	<u>Type mismatch</u>
123	<u>Undefined dialog record field</u>

124 Unknown Command, Subroutine, or Function
125 Unexpected end of macro
126 WEND without WHILE
127 Wrong number of dimensions
129 Too many nested control structures
130 SELECT without END SELECT
131 Illegal REDIM to dialog record
132 External call caused string overflow
133 Wrong number or type of arguments for DLL call
134 An argument to a function contained an illegal date or time.
137 The specified path is not a valid path option.
138 The current selection cannot be modified by this command.
139 Only one user dialog may be up at any time.
140 Dialog control identifier does not match any current control.
141 The () statement is not available on this dialog control type.
142 Specified application is not currently running
143