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

(*)		or .Replace
Nonbreaking hyphen (–)	"^~"	.Find or .Replace
Optional hyphen (¬)	"^-"	.Find or .Replace
White space [2]	"^w"	.Find

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

Note

([1]) 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".

([2]) 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_ChangeRulerMode
WW2_CountMenuItems()
WW2_EditFind
WW2_EditFindChar
WW2_EditReplace
WW2_EditReplaceChar
WW2_FileFind
WW2_FileTemplates
WW2_Files$()
WW2_FootnoteOptions
WW2_FormatBordersAndShading
WW2_FormatCharacter
WW2_FormatDefineStyleChar
WW2_GetToolButton()
WW2_GetToolMacro$()
WW2_Insert
WW2_InsertFootnote
WW2_InsertIndex
WW2_InsertSymbol
WW2_InsertTableOfContents
WW2_KeyCode
WW2_MenuMacro$()
WW2_MenuText$()
WW2_PrintMerge
WW2_PrintMergeCheck
WW2_PrintMergeCreateDataSource
WW2_PrintMergeCreateHeaderSource
WW2_PrintMergeHelper
WW2_PrintMergeSelection
WW2_PrintMergeToDoc
WW2_PrintMergeToPrinter
WW2_RenameMenu
WW2_RulerMode
WW2_TableColumnWidth
WW2_TableRowHeight
WW2_ToolsHyphenation
WW2_ToolsMacro
WW2_ToolsOptionsGeneral
WW2_ToolsOptionsKeyboard
WW2_ToolsOptionsMenus
WW2_ToolsOptionsPrint
WW2_ToolsOptionsToolbar
WW2_ToolsOptionsView
WW2_ToolsRevisionsMark
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 he 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 to 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 a 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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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

¬Example

ClearFormField

Clears 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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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()

¬Example

CountDocumentVars()

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

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

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 (), 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

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

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	Specifies whether to display the header or footer area. The possible values of .Type depend on the settings of .FirstPage and .OddAndEvenPages. If both .FirstPage and .OddAndEvenPages are set to 0 (zero): 0 (zero) Header 1 Footer If .FirstPage is set to 1 and .OddAndEvenPages is set to 0 (zero): 0 (zero) Header 1 Footer 2 First header 3 First footer If .FirstPage is set to 0 (zero) and .OddAndEvenPages is set to 1: 0 (zero) Even header 1 Even footer 2 Odd header 3 Odd footer If both .FirstPage and .OddAndEvenPages are set to 1: 0 (zero) First header 1 First footer 2 Even header 3 Even footer 4 Odd header 5 Odd footer
.FirstPage	If 1, allows a header or footer for the first page that differs from the rest of the pages in the section.
.OddAndEvenPages	If 1, allows one header or footer

	for even-numbered pages and a different header or footer for odd-numbered pages.
.HeaderDistance	The distance from the top of the page to the header.
.FooterDistance	The distance from the bottom of the page to the footer.

See also

View Statements and Functions

FilePageSetup

ViewFooter

ViewHeader

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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 "!Employe.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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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$()

¬Example

SelectionFileName$()

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 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)	Concatenates two strings. For example, the result of "Microsoft " + "Word" is "Microsoft Word". You must ensure that spaces are included in the strings being concatenated to avoid running words or characters together. 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"). Note that you also use + as the addition operator.

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	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: True AND True — True False AND True — False True AND False — False False AND False — False
OR	If either or both expressions evaluate true, the result is true. The result is determined as follows: True OR True — True False OR True — True True OR False — True False OR False — False
NOT	The result is determined as follows: NOT False — True NOT True — False 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).

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

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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 AppActivate.

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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 AppMaximize "Microsoft Excel", 1 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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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$*, see AppActivate.
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance from the upper-left corner of the screen to the upper-left corner of the application window or icon, in points (72 points = 1 inch). Negative measurements are allowed only if you specify *WindowName$*.

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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

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

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

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

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

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

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

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

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

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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$* can be a document filename by itself, provided the filename extension is registered in the [Extensions] section of the WIN.INI file. **Shell** starts the associated application and opens the document. To display an MS-DOS window, specify Environ$ ("COMSPEC") 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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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
.SmartQuotes	If 1, Word inserts "smart" quotation marks (" " and ' ') and apostrophes (').
.InitialCaps	If 1, Word corrects words in which the first two letters are capitalized. For example, "WOrd" becomes "Word."
.SentenceCaps	If 1, Word capitalizes the first letter of new sentences.
.Days	If 1, Word capitalizes the days of the week. For example, "tuesday" becomes "Tuesday."
.ReplaceText	If 1, activates automatic replacement of text.
.Formatting	If 1, formatting is stored with the replacement text when a replacement entry is added; available only if text is selected before running **ToolsAutoCorrect**.
.Replace	The text you want to replace automatically with the text specified by .With (for example, a person's initials).
.With	The text you want to insert automatically when the text specified by .Replace is typed (for example, a person's full name).
.Add	Adds the text specified by .Replace and .With to the list of replacement entries.
.Delete	Deletes the replacement entry specified by .Replace.

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

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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()

¬Example

CountBookmarks()

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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

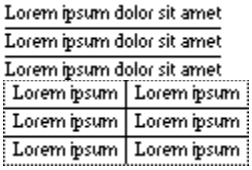
ShadingPattern

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.



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

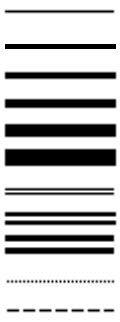
BorderLineStyle, BorderLineStyle()

BorderLineStyle *Style*

BorderLineStyle()

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

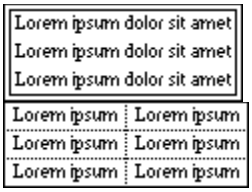
ShadingPattern

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.



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	If the selection consists of more than one of the following items, specifies to which item or items the border format is applied: 0 (zero) Paragraphs 1 Graphic 2 Cells 3 Whole table If .ApplyTo is omitted, the default for the selection is assumed.
.Shadow	Specifies whether to apply a shadow to the border of paragraphs or a graphic: 0 (zero) Does not apply a shadow. 1 Applies a shadow. 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.
.TopBorder, .LeftBorder, .BottomBorder, .RightBorder	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).
.HorizBorder	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.
.VertBorder	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.)
.TopColor, .LeftColor, .BottomColor, .RightColor, .HorizColor, .VertColor	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).
.FineShading	A shading pattern in the range 0 (zero) to 40 corresponding to a shading percentage in 2.5 percent increments. If .FineShading is anything but 0 (zero), .Shading is ignored.
.FromText	The distance of the border from adjacent text, in points or a text measurement. Valid only for paragraphs; otherwise, .FromText must be an empty string ("") or omitted or an error will occur.
.Shading	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).
.Foreground	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).
.Background	The color to be applied to the background of the shading, in the range from 0 (zero), which is Auto, through 16.
.Tab	Specifies which tab to select when you display the Borders And Shading dialog box with a **Dialog** or **Dialog()** 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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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.
.MoveWithText	If 1, the frame moves as text is added or removed around it.
.LockAnchor	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.
.RemoveFrame	Removes the frame format from the selected text or graphic.

See also

Borders and Frames Statements and Functions

FormatDefineStyleFrame

InsertFrame

RemoveFrames

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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 13 1 14 2 15 3 16 4 17 5 18 6 19 7 20 8 21 9 22 10 23 11 24 12 25

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

ViewBorderToolbar

ViewBorderToolbar

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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**
    Series of instructions]
End If

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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
.Remove	Removes bullets or numbering from the selection.
.Hang	If 1, applies a hanging indent to the selected paragraphs.
.Preset	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: 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

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

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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()

¬Example

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, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+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

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

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

Date and Time Functions

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

Disk Access Statements and Functions

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

Environment Statements and Functions

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

Menu Customization Statements and Functions

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

Selection Statements and Functions

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

String Functions

CleanString$()	Changes nonprinting characters and special Word characters to spaces (ANSI character code 32)
DOSToWin$(), WinToDOS$()	Translate a string from the original equipment manufacturer (OEM) character set to the Windows character set, and vice versa
LTrim$(), RTrim$()	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

CountDocumentVars(), GetDocumentVar$(), GetDocumentVarName$(), SetDocumentVar	Manage document variables.
FileNameInfo$()	Returns all or part of the path and filename of the specified file.
GetPrivateProfileString$(), SetPrivateProfileString	Store values in private settings files; retrieve values from private settings files.
IsTemplateDirty()	Returns a value indicating whether the active template has changed since it was last saved. Note that **IsDirty()** has changed to **IsDocumentDirty()**.
PathFromMacPath$()	Converts a Macintosh path and filename to a valid path and filename for the current operating system.
SaveTemplate	Saves changes to the active template or the global template.
ScreenUpdating	Controls whether changes are displayed on your monitor while a macro is running.
SelectionFileName$()	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 (\).
SetTemplateDirty	Controls whether Word recognizes a template as changed since the last time the template was saved. Note that **SetDirty** has changed to **SetDocumentDirty**.
SortArray	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.
Stop	Now includes an argument to suspend the macro without displaying an error message.
WaitCursor	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
File*Number*
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
Overtype, Overtype()
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
GoToNext*Item*
GoToPrevious*Item*
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
ShowHeading*Number*
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

Window*Number*

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	Illegal function call
6	Overflow
7	Out of memory
9	Subscript out of range
11	Division by zero
14	Out of string space
22	Invalid array dimension
24	Bad parameter
25	Out of memory (stack space)
26	Dialog needs End Dialog or a push button
28	Directory already exists
39	CASE ELSE expected
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
62	Input past end of file
64	Bad file name
67	Too many files
74	Rename across disks
75	Path/File access error
76	Path not found
100	Syntax error
101	Comma missing
102	Command failed
103	Dialog record variable expected
104	ELSE without IF
105	END IF without IF
109	INPUT missing
111	Expression too complex
112	Identifier expected
113	Duplicate label
114	Label not found

115	Right parenthesis missing
116	Argument-count mismatch
117	Missing NEXT or WEND
118	Nested SUB or FUNCTION definitions
119	NEXT without FOR
120	Array already dimensioned
122	Type mismatch
123	Undefined dialog record field
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	The dialog control with the focus may not be disabled or hidden.
144	Focus may not be set to a hidden or disabled control.
149	The () command cannot be called as a function.
150	Dialog control identifier is already defined.
152	This command is not available because no document is open.
155	The selection does not start in a field.
157	The field cannot contain data.
158	The value of one of the fields is too low.
159	The value of one of the fields is too high.
160	Wrong number of parameters
161	Cannot change dialogs when focus is changing (action 4)
162	The () command can only be called as a function.
163	This statement can only be used when a user dialog is active.
164	Array variable has not been initialized.
500	Cannot initiate link
501	Invalid channel number
502	Application does not respond
503	Process failed in other application

504 Window does not exist
505 Cannot activate application
506 Cannot send keys
508 Other application is busy
509 The () command is unavailable because ().
511 No such macro or command
512 Value out of range
513 String too long
514 Document not open
528 Unable to load spelling checker
529 Cannot open dictionary
530 Dialog box description too complex
535 Macro cannot be run because it is already running.
536 There is no macro with that name.
537 Unable to run macro specified
538 Unable to edit macro specified
539 Unable to rename macro specified
540 Unable to delete macro specified
541 Unable to set description of macro specified
543 Unable to open specified library
544 Unable to execute the scroll command; the scrollbar is not active.
545 The () statement is currently disabled.
546 Footnotes or endnotes must start at 1 if numbering is not continuous.
547 Network Permission Error
549 The specified menu or menu item does not exist.
551 Word is unable to perform this action because the specified template is locked.
552 Word is unable to perform this action because the specified template does not exist.
553 Unable to create macro specified
554 No drawing range has been set.
555 The bookmark specified for the drawing range is invalid.
556 Wrong drawing object type for this command
557 Could not insert the drawing object
558 At least one subdocument in this master document is locked. No changes can be made to any locked subdocuments.
559 The current selection is a block.
560 The revision marks are not visible.
561 Document is protected
562 ToolsGrammarStatisticsArray cannot be run on a document that contains more than one language format.
563 The document is not a master document.
564 There are no subdocuments in that direction.
565 The specified document is not in the Add-in list.
566 The specified Word library cannot be unloaded because it is in use.

567	Cannot add more than 25 items in the dropdown list box
568	The specified font doesn't exist
569	PatternMatch and SoundsLike parameters cannot both be set to 1
570	Cannot sort arrays with more than two dimensions

Word Error Messages

Error #	Message
1001	There is insufficient memory. Save the document now.
1003	You can specify only one line, footnote, endnote, annotation, or field at a time.
1005	This bookmark does not exist.
1006	You entered multiple destinations for a page, line, footnote, endnote, or annotation.
1008	Word cannot insert a section break here.
1009	The bookmark name is not valid.
1011	There is not enough memory to compile the index.
1013	There is not enough memory to run the DDE application.
1014	There is not enough memory to run the application.
1016	There is not enough memory to complete the operation.
1017	There is not enough memory to update the display.
1018	There is not enough memory to define the AutoText entry.
1019	There is not enough memory to merge the styles.
1020	There is not enough memory to display the outline.
1021	There is not enough memory to display the ruler.
1022	The name you typed is not a valid AutoText entry. Use the AutoText button on the Standard toolbar to define AutoText entries that can be inserted as a long piece of text or a graphic.
1023	There is a serious disk error on file ().
1024	The file () is not available.
1025	Word cannot open the document.
1026	This style name does not exist.
1027	There is insufficient memory, and Word cannot perform the replace operation.
1028	The search item was not found.
1029	There is not enough memory to display or print the picture.
1030	The dimensions after cropping are too small or too large.
1031	The dimensions after resizing are too small or too large.
1032	The file is too large to save. Delete some text and try again.
1033	Word cannot use the DOT-PATH specified in the WINWORD6.INI file because it is not valid.
1034	Word cannot use the INI-PATH specified in the WINWORD6.INI file because it is not valid.
1035	Word cannot use the TOOLS-PATH specified in the WINWORD6.INI file because it is not valid.
1036	Word cannot start the converter ().
1037	There is not enough memory to run this converter.
1038	The password is incorrect. Word cannot overwrite the document.
1039	This is not a valid hyphenation zone measurement.
1041	This document template does not exist.
1042	Settings you chose for the left and right margins, column spacing, or paragraph indents are too large for

the page width in some sections.

1043 This tab stop is too large.

1044 Word cannot print. There is no printer installed.

1045 This is not a valid print range.

1046 Windows cannot print due to a problem with the current printer setup.

1047 There is not enough memory to repaginate or print this document.

1048 Windows needs more disk space to print this document.

1049 Another window cannot be opened until one is closed.

1050 This is not a valid number.

1051 This is not a valid measurement.

1052 The number must be between () and ().

1053 The measurement must be between () and ().

1054 Word cannot write to file ().

1055 The document name or path is not valid.

1056 This is not a valid filename.

1057 Word cannot give a document the same name as an open document.

1058 You cannot save a template file to non-template format.

1059 This file is read-only.

1060 Word cannot save or create this file. Make sure the disk is not write protected

1062 Fields are nested too deeply.

1066 There is not enough memory to run Word.

1069 Word cannot open the existing ().

1070 This is not a valid date.

1071 This is not a valid style name.

1072 The style sheet is full. Word cannot define the new style.

1073 You cannot base a style on itself.

1074 The Based On style name does not exist or is of incorrect type.

1075 The Next style name does not exist or is of incorrect type.

1076 Word cannot merge the style sheet from the active template.

1077 This style name already exists or is reserved for a built-in style.

1078 This file could not be found.

1079 There is not enough memory for such a large Clipboard.

1080 The indent size is too large.

1081 The paragraph is too wide.

1083 The command name must have an extension.

1084 Word cannot open this document template.

1085 The document template is not valid.

1090 The style sheet is full. The style of some paragraphs may become Normal.

1091 Word cannot insert a file into itself.

1093 There is not enough memory to run the thesaurus.

1094 Word cannot start the thesaurus.

1095 There is not enough memory to run the spelling checker.

1097 Word cannot start the spelling checker.

1103 Word cannot find the file WORDCBT.CBT.
1104 This is not a valid tab stop.
1105 There are too many tab stops set in this paragraph.
1106 There are too many tab stops to clear at one time.
1107 Word found no XE (Index Entry) fields for the index.
1108 Word cannot create a work file.
1109 The WINWORD.OPT file is not valid. Word will use the defaults.
1111 Word cannot open a window for the result. Close some open windows and try again.
1112 You cannot insert DATA, NEXT, NEXTIF, or SKIPIF fields inside other fields.
1113 You cannot include DATA, NEXT, NEXTIF, or SKIPIF fields in annotations, headers, footers, footnotes or endnotes.
1114 A DATA field must precede NEXT, NEXTIF, or SKIPIF fields.
1115 A DATA field does not contain a data filename.
1116 There is a syntax error in a field condition.
1117 Word does not recognize the filename in the data source you typed.
1118 Word does not recognize the filename in the header source you typed.
1119 The requested record is beyond the end of the mail merge data source.
1120 There is a printer error.
1121 Word cannot change printers. No printers are installed.
1122 The number in the Start At box is not valid.
1123 The number in the Format box is not valid.
1124 The numbers in the Start At and Format boxes are not valid.
1125 Word found no TC (Table of Contents Entry) fields for the table of contents.
1126 Word found no paragraphs with heading styles to include in the table of contents.
1127 Make a selection first.
1128 This is not a valid selection.
1129 The document is too large to save. Delete some text before saving.
1130 The document is too large for Word to handle.
1132 There is insufficient memory. Close an application.
1133 This search list is not valid.
1134 This search expression is not valid.
1138 The window is too small.
1140 This is not a valid action for endnotes.
1142 Word found no revision marks.
1143 This document has too many styles. Word discarded some styles.
1144 Word cannot read the formatting in this document.
1145 Word found no footnotes.
1146 Word found no annotations.
1147 There is insufficient memory. The fonts in the copied text may be incorrect.
1151 Word cannot move text containing a section break to the selected destination.
1152 Word cannot move footnote, endnote, or annotation references to the selected destination.
1153 Word cannot replace footnote, endnote, or annotation references.
1154 There are too many edits in the document. This operation will be incomplete. Save your work.

1155 This is not a valid action for the end of a row.
1156 This is not a valid action for footnotes.
1157 Word did not save the document.
1158 The original file may be damaged due to a serious disk error. Save it with a new filename.
1159 You are working without a Word work file and memory is nearly full. Save your work.
1160 The Word work file and memory are nearly full. Save your work.
1161 This exceeds the maximum number of columns.
1162 This exceeds the maximum width.
1163 The end of a row cannot be deleted.
1164 You cannot insert this selection into a table.
1170 You cannot paste this selection into a table.
1172 Word cannot paste text containing a section break.
1175 The Paste command failed because the copy and paste areas are different shapes.
1176 There is insufficient memory. Close extra windows and save your work.
1177 The document name or path is not valid.
1179 This is not a valid action for annotations.
1180 The value is out of range.
1181 Word could not create the work file. Check the temp environment variable.
1182 There is not enough memory to sort.
1183 Word cannot sort fields in the selection.
1184 Word found no valid records to sort.
1185 The style you want to create is based on too many styles.
1186 There is insufficient memory. Word is closing the saved document.
1190 There is an unrecoverable disk error on file ().
1191 The disk is full trying to write to (). Free some space on this drive or save the document on a different disk.
1192 This macro is too big for Word to run. Split it into two or more macros.
1193 The macro () in the template () is too big and will be truncated. To completely convert the macro, edit it in Word for Windows 1.x and split it into two or more macros.
1194 You cannot close a running macro.
1195 () is not a valid macro or command name.
1196 There is not enough memory to record the command.
1198 Cannot run Word: incorrect Windows or MS-DOS version.
1200 Word cannot start Examples and Demos.
1202 There is not enough memory to run Examples and Demos.
1208 Word cannot rename or delete a macro that is open for editing.
1209 You cannot edit a macro while it is being recorded.
1210 This macro line is too long.
1211 You cannot record over a macro that is open for editing.
1215 There is insufficient memory. Close extra windows and try again.
1217 One or more rows are too wide to split.
1218 The recorded macro was too long and has been truncated.
1219 One or more rows are too wide for this operation.

1220 Word cannot find or run the application.

1225 You cannot copy or move this selection.

1226 There is insufficient memory. Word cannot display the requested font.

1227 Word cannot open the graphics file.

1229 This style has a circular Based On list.

1234 Records with an incorrect number of fields will be skipped.

1235 There is not enough memory to complete the operation.

1236 There is insufficient memory. The list may be incomplete.

1237 You cannot quit Microsoft Word because a dialog is active. Switch to Microsoft Word first and close the dialog.

1240 You cannot repaginate until a printer is installed.

1241 Word for Windows is opening a Word for OS/2 document. (This document may be fully converted by saving it as a Word for Windows document from within Word for OS/2.)

1245 Word cannot display this picture format.

1246 The specified data type is unavailable.

1247 Word cannot start the file converter () because it is being used by another Word session.

1248 Word cannot locate the server application for () objects. Install the server application with the Setup program.

1249 Word cannot obtain the data for the () link.

1250 Word cannot change the function of the specified key.

1251 The key name is not valid.

1255 The spelling checker is in use.

1256 The link does not exist.

1257 You cannot save while the file is in use by another process. Try saving the file with a new name.

1260 Word cannot save this file because it is already open elsewhere.

1261 Word cannot open this file because it is being updated by another process.

1262 You cannot modify a specified link.

1263 You have tried to open too many custom dictionaries. Word can have up to 10 custom dictionaries open at the same time.

1264 The server application, source file, or item cannot be found. Make sure the application is properly installed, and that it has not been deleted, moved, or renamed.

1265 You cannot change printer orientation with the current printer installed.

1266 Line spacing must be at least ().

1267 Word cannot create the custom dictionary ().

1268 Word cannot open custom dictionary ().

1269 Word cannot edit the ().

1270 Word cannot use the AUTOSAVE-PATH specified in the WINWORD6.INI file because it is not valid.

1271 Word cannot recognize this language.

1272 There is not enough memory to run Help for WordPerfect Users.

1273 Word cannot start Help for WordPerfect Users.

1275 The object () is locked for editing.

1276 No text or formatting is in the Find box.

1277 The custom dictionary is full. The word was not added.

1278 The custom dictionary () is too large. Try dividing it in half.

1279 A spelling checker error occurred. Word is ending the current session.
1280 The custom dictionary () is not available.
1282 The number in the Position box must be positive.
1283 This file was not recognized by the specified graphics filter.
1284 The graphics file is too large to be converted.
1285 This graphics file may be damaged and cannot be converted.
1286 The graphics filter was unable to convert this file.
1287 Word cannot start the graphics filter.
1289 The grammar checker is in use.
1290 There is not enough memory to run the grammar checker.
1291 Word cannot start the grammar checker.
1292 Word cannot read this file. Graphics filters must be installed in the WIN.INI file.
1293 Word does not recognize the object ().
1295 The item name () 000() () cannot be found.
1297 The setting does not exist or cannot be deleted.
1298 This document could not be registered.
1299 The list of paths is full.
1300 Word cannot save the document while the backup file is open. Clear the backup option or save the document with a new name.
1302 The operation is cancelled.
1304 There is not enough memory to hyphenate the document.
1305 Word cannot locate the hyphenator file.
1306 The file viewer could not be initialized.
1309 There is not enough memory to create the hand annotation.
1310 Make the list separator and the decimal separator different before calculating.
1311 The thesaurus is in use.
1312 The password is incorrect. Word cannot open the document.
1313 Word cannot make the current document into both a mail merge main document and a data document.
1314 The disk drive is not valid.
1315 The path is not valid.
1316 This action will not change the default page number format. To change the default page number format for this section, use Insert Page Number and choose the Format button.
1317 The disk is full.
1318 There is not enough memory to display the Mail Merge toolbar.
1319 The disk search failed. Word has restored the previous search path.
1320 The setting could not be created.
1321 The directory is not valid.
1323 Word cannot find the () ().
1324 Word cannot find the () () for ().
1325 The style name contains a character that is not valid.
1326 The style name is too long. The maximum number of characters allowed is 253.
1327 Word cannot create the specified destination file.
1328 This file is empty or does not contain a graphic image.

1331 A grammar checker error occurred. Word is ending the current session.
1332 The path cannot contain any periods as path elements.
1333 Word is unable to start Help.
1334 You cannot edit the result of a link or embedded object.
1335 Word cannot convert graphics files because no printer has been selected in File Print.
1336 Word cannot create the () file because no field names were specified.
1338 The maximum number of fields allowed is 31.
1339 Word cannot write to the WIN.INI file.
1340 The destination document did not accept the update.
1341 This filename is not valid or it contains a path.
1342 Graphics files can be inserted using the Picture option from the Insert menu.
1343 Word is hyphenating another document.
1345 You cannot print a graphics file using the Find File command.
1346 You cannot attach a document template as a data or header source.
1347 A hyphenation error occurred. Word is ending the current session.
1348 A thesaurus error occurred. Word is ending the current session.
1349 Word cannot use the DOC-PATH specified in the WINWORD6.INI file because it is not valid.
1351 Word cannot search for paragraph marks in Outline View.
1352 Word cannot delete the old backup file because it is read-only or another user has it open.
1353 Word cannot open the () () for ().
1354 There are too many characters in the selection for the edit text window.
1355 The selection contains characters that cannot be edited using the edit text window.
1356 Word has finished searching the (). The search item was not found.
1357 Word has finished searching the ().
1358 Word cannot use the PIC-PATH specified in the WINWORD6.INI file because it is not valid.
1359 The directory name is not valid.
1360 You do not have network permissions for this action.
1361 This file is in use by another application or user.
1362 There are too many open files.
1363 Cannot delete the current working directory.
1364 A file error has occurred.
1366 There is no picture available at the specified location.
1367 A directory of the proposed name already exists.
1368 A file of the proposed name already exists.
1369 Word cannot rename the file.
1371 You must type some text for a custom footnote or endnote mark.
1372 The AutoText name is not valid.
1373 The AutoText name already exists.
1374 The macro name is not valid.
1375 The macro name already exists.
1376 The organizer was unable to rename the () ().
1377 The built in () () cannot be deleted.
1378 A built in style cannot be renamed.

1380 The password is too long.

1381 Word detected an error while trying to bring up the Connect Network dialog box.

1382 There is not enough memory to edit a macro.

1383 You cannot close Microsoft Word because a dialog is active. Switch to Microsoft Word first and close the dialog.

1385 There is not enough memory to convert the macros in this file.

1386 The organizer was unable to copy the () ().

1387 Word could not determine the destination.

1388 Word cannot add the entry because AutoCorrect entries cannot be blank.

1389 The password is incorrect.

1391 Word cannot complete the save due to a file permission error.

1392 Unable to add because AutoCorrect entries are limited to 31 characters in length.

1394 Unable to add entry to AutoCorrect because there is insufficient memory.

1395 AutoCorrect cannot replace text which contains a space character. Please remove the space, or replace it with something else.

1396 Word could not save the language setting in the custom dictionary.

1397 The active document is not a valid mail merge main document.

1398 Word did not update the source file.

1399 Dropdown list form fields must have at least one item in their list.

1400 Check the floppy drive to make sure the door is closed and it contains the correct disk.

1402 There is not enough memory to save the entire Ignore All list.

1405 The document is locked and cannot be opened.

1406 The disk is full or too many files are open.

1407 Word cannot replace the found item with text containing a break.

1408 This version of the spelling checker does not understand the language formatting of custom dictionary ().

1409 Column widths cannot be less than ().

1410 One or more columns or column spaces with zero width were found and removed.

1411 There must be at least one column defined.

1412 The linked document in () is unavailable.

1413 This font style name does not exist.

1414 You cannot put drawing objects into a text box, callout, annotation, footnote, endnote, or macro.

1415 The requested envelope size is not valid.

1419 The page height for some sections is greater than the maximum allowable page height.

1420 The top/bottom margins are too large for the page height in some sections.

1422 Increasing left margin causes first column to become too narrow in some sections having unevenly spaced columns.

1423 Increasing right margin causes last column to become too narrow in some sections having unevenly spaced columns.

1424 The network drive cannot be connected because there are no free drive letters.

1425 The network drive cannot be accessed without a password.

1426 The network path is not valid.

1427 The network password is not valid.

1428 Word has finished searching the (). () replacements were made.

1429 Word has finished searching the (). () replacement was made.

1431 () is not a valid () dll.
1432 () is an incorrect version of the () DLL.
1433 There is not enough memory to make the indicated changes to the menu.
1434 Word cannot find the designated menu.
1435 There is no header layer for this page. All available layers will be displayed.
1436 The number must be between () and ().
1437 This caption label is not valid.
1438 You can have no more than 25 items in your drop-down list box.
1439 Word could not merge these documents or insert this database.
1440 This command is available only from a macro.
1441 This AutoCorrect entry does not exist.
1442 Word could not shrink the document by one page because the document is already only one page.
1443 After several attempts, Word was unable to shrink the document by one page.
1444 Word could not successfully convert the picture metafile into drawing objects.
1445 There are not enough system resources to complete the operation.
1446 The new toolbar must have a name.
1447 The formula is empty.
1448 The default directory for this item cannot be on a floppy drive. Please choose a directory on a hard drive.
1449 That toolbar already exists in the specified context. Please choose a different name.
1450 The document contains a character set that is not supported by this version of Word.
1451 Word found no TA (Table of Authorities Entry) fields for the table of authorities.
1452 Word cannot open () as a data or header source because it contains no data.
1453 The name you have chosen is already used for a built-in toolbar. Please choose a different name.
1454 Word cannot save the mail file.
1455 The new menu name cannot be the same as an existing menu name.
1456 The selection does not consist of heading levels.
1457 An unlocked subdocument already exists in another master document.
1458 The file cannot be opened with write privileges.
1459 The master document needs to be saved. Please save it now and try the operation again.
1460 Word could not save the subdocument. Please try a full save of the master document.
1461 Unknown mail address.
1462 Ambiguous mail address. There are two or more people whose mail addresses match the one specified.
1463 Word was unable to mail your document.
1464 The Find What text contains a Pattern Match expression which is not valid.
1465 The field delimiter cannot be the same as the record delimiter.
1466 The row or column value is too large or small for this sheet of labels.
1467 The margins, label size, and number across or down values produce a page that is larger than the label page size.
1468 Horizontal and vertical pitch must be greater than or equal to the label width and height, respectively.
1469 Label width and height must be positive values.
1470 Word could not recognize the field and record delimiter you specified.
1471 This save format is not valid.
1472 Word cannot save a document as form data only with the same name as an open document.

1473 The MACRO field macro () is not available.

1474 Word cannot print because there is no default printer selected. Please select a printer.

1475 Your mail system does not support certain services needed for document routing.

1476 Too many mail sessions. Log off other mail sessions and try again.

1477 General mail failure. Close Microsoft Word, restart the mail system, and try again.

1478 The entry is empty.

1479 No citation was found.

1480 The Update Source command cannot be used to update a master document.

1481 Word cannot rename a file to another drive.

1482 The Find What text for a Sounds Like search can only contain non-accented alphabetic letters.

1483 Word found no captions to include in the table of figures.

1484 Word cannot locate its help file (WRDHLP32.DLL). You can install the help file with Word's Setup program.

1485 The operation cannot be completed because a master document cannot contain more than 9 levels of subdocuments.

1486 There has been a network or file permission error. The network connection may be lost.

1487 Too many files are open. Please close a window.

1488 Too many DDE channels are open. Please close a window.

1489 Too many Word documents are open. Please close a window.

1490 The template cannot be customized because it is locked for editing.

1491 The style () refers to more than one style in the destination document.

1492 This menu item is automatically added by Word and cannot be removed from the menu.

1493 A proofing tool menu item is automatically added when Word detects that the tool has been installed. This item cannot be removed from the menu.

1494 The Find What text contains a range that is not valid.

1495 This command supports only table and paragraph-delimited lists.

1496 Record () contained too many field delimiters (()).

1497 Record () contained too few field delimiters (()).

1498 Word found no endnotes.

1499 Word Add-in could not be loaded.

1500 Start At must be between () and () for this format.

1502 Word cannot attach a document to a protected template.

1503 You cannot record a macro to a locked file.

1504 No index entries were marked.

1505 Word cannot insert a database that has no field names selected.

1506 Word cannot update the index.

1507 Subdocument () cannot be inserted because it is the current master document or because it contains the current master document as a subdocument.

1508 The maximum length value must be equal to or greater than the length of the default text, () characters.

1509 Word cannot create a subdocument within a field code or result.

1510 Word cannot create a subdocument within a table.

1511 Word cannot create a subdocument within a frame.

1512 () is not a style name.

1513 () refers to more than one style.

1515 Word cannot copy style () because it matches a style of a different type.
1516 Word cannot display Macintosh pictures without a graphics filter.
1517 Word cannot find the designated menu item.
1518 The macro () cannot be renamed or deleted because it is currently open for editing or recording.
1519 Word cannot load the Button Editor, COMMTB.DLL.
1520 Mail is not installed on your system.
1521 Word has already routed this message. Reset the routing slip and try again.
1522 Word cannot run a macro until it has finished routing this document.
1523 The document does not have a routing slip. Add a routing slip and try again.
1524 Word for Windows cannot open a Word for the Macintosh glossary file.
1525 Word did not update the source file because it is protected or is an open subdocument.
1526 You must type a character for the custom separator.
1527 The Replace With text contains a group number which is out of range.
1528 () is not a valid special character for the Replace With box.
1529 () is not a valid special character for the Find What box.
1530 The current selection does not contain a valid table or list.
1531 Word cannot read the header or footer in this document. Save the document in Rich Text Format, close it, and then reopen it.
1533 The list of filenames is too long. Word cannot open this many files at one time.
1534 Word cannot merge documents that can be distributed by mail or fax without a valid mail address. Choose the Setup button to select a mail address data field.
1535 Word could not merge the main document with the data source because the data records were empty or no data records matched your query options.
1536 Word cannot find or run the Print Manager.
1537 A valid date or time is required.
1538 A valid number is required.
1539 Word found no paragraphs with the styles needed to compile a table of figures.
1540 Word found no paragraphs with the styles needed to compile a table of contents.
1541 Word found no TC (Table of Contents Entry) fields for the table of figures.
1542 Word could not parse your query options into a valid SQL string.
1543 You can type no more than 32 characters in the Text Before and Text After boxes.
1544 Word could not re-establish a DDE connection to () to complete the current task.
1545 This is not a valid filename '().
1546 Word could not retrieve a SQL string from the active data source.
1547 This document cannot be opened in the Organizer dialog box.
1548 The default menu for this command does not exist. Use the Customize command to add this command to a menu.
1549 Word has increased the envelope height or width because it was too small. The minimum envelope dimensions are 6.4 by 2.13 inches.
1550 This toolbar name is not valid.
1551 This toolbar name already exists.
1552 The built-in styles Normal and Default Paragraph Font cannot be based on any style.
1553 Word could not open () because it didn't contain a valid database
1554 This command is unavailable because the form field was not inserted with the Forms toolbar or by using

the Insert Form Field dialog box.

1555 Word cannot switch applications.

1556 Word could not locate or start ().

1557 The organizer could not delete the () ().

1558 The () () does not exist.

1559 Word could not restore an automatically saved document from the last session.

1560 Word could not replace the selection with the specified database.

1561 Word cannot start the grammar checker.

1562 No citations were marked.

1563 Word cannot open () as a data or header source because it is a mail merge main document.

1564 The document cannot be locked.

1565 You cannot send a catalog created by merging documents directly to mail, fax, or a printer.

1566 Word could not merge these documents or insert this database.

1567 Microsoft Mail Local Fax is not installed on your system.

1568 The Windows registration database file is not valid. You can correct it with Word's Setup program.

1569 Word cannot load the OLE 2.0 or Docfile libraries.

1570 There is not enough available disk space to run Word.

1571 You cannot paste form fields into annotations, headers, footers, footnotes, endnotes or text boxes.

1572 The formatting in this document is too complex. Please full save the document now.

1573 The undo operation is prevented by a lock on a subdocument or master document.

1574 The operation is prevented by a lock on a subdocument or master document.

1575 A valid date format is required.

1576 A valid number format is required.

1577 The Add-in template is not valid.

1578 This key name is not valid for sorting. Please type or select another key from the list.

1579 Word cannot sort this table or selection because all of its rows are table headings.

1580 Word cannot recognize these numbers because of the list separator, number format or currency format settings used in Windows. To see the current settings, double-click the International icon in the Windows Control Panel.

1581 Word cannot start Quick Preview.

1582 Cannot go to requested bookmark.

1583 The complexity of your query options generated a SQL string that was too long for Access 1.0 to handle. Please upgrade to a later version of Microsoft Access to perform query options.

1584 Word cannot sort Microsoft Access queries. You will need to sort the query in Microsoft Access.

1585 Normal.dot is already open as an Add-in.

1586 Word is opening a Word for Windows 1.x template. Macros from Word for Windows 1.x files cannot be converted and will be lost if this file is saved in Word for Windows 6.0 format.

1587 There are too many drawing objects on the page to update the display.

1588 You cannot put section breaks into a header, footer, footnote, endnote, annotation, text box, callout, or macro.

1589 The operation cannot be completed because the subdocument is open in another window.

1590 The operation cannot be completed because the subdocument has a higher protection rating than the master document. Upgrade the protection rating of the master document using Tools Protection.

1591 The operation cannot be completed because the master or subdocument has open embedded objects in it.

Update and close these objects.

1592 Word cannot make the requested network connection.

1593 There are too many characters in the form field.

1594 The selection is marked as deleted text.

1595 Word cannot search for paragraph marks in Master Document View.

1596 () is not a valid special character for the Find What box or is not supported when the Use Pattern Matching checkbox is selected

1597 Not enough memory to initialize the menus. Save all changes and close Word.

1598 You cannot edit a picture which is not saved in the document.

1599 The selected drawing objects will not fit in a single group.

1600 This action is not available because the AutoFormatter dialog is running.

1601 Word could not insert the database because your query options produced an empty result.

1602 Word cannot open () as a header file because it cannot be converted to a native Word file.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

AllCaps Example

This example applies the all caps format to the entire selection if part of the selection is formatted as all caps:

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

AllCaps, AllCaps()

¬Example

AllCaps [*On*]

AllCaps()

The **AllCaps** statement adds or removes the all caps character format for the current selection, or controls all caps formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the all caps format: 1 Formats the selection as all caps 0 (zero) Removes the all caps format Omitted Toggles the all caps format

The **AllCaps()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as all caps
-1	If part of the selection is formatted as all caps
1	If all the selection is formatted as all caps

See also

Character Formatting Statements and Functions

FormatFont

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Bold Example

This example applies the bold format to the entire selection if part of the selection is formatted as bold:

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

Bold, Bold()

¬Example

Bold [*On*]

Bold()

The **Bold** statement adds or removes the bold character format for the current selection, or controls bold formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the bold format: 1 Formats the selection as bold 0 (zero) Removes the bold format Omitted Toggles the bold format

The **Bold()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as bold
-1	If part of the selection is formatted as bold
1	If all the selection is formatted as bold

See also

Character Formatting Statements and Functions

FormatFont

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CharColor Example

This example selects the current paragraph (using the predefined bookmark "\Para") and applies magenta character color formatting if it contains the string "Comments: ".

```
EditGoTo "\Para"
If InStr(Selection$(), "Comments: ") Then CharColor 5
```

CharColor, CharColor()

¬Example

CharColor *Color*

CharColor()

The **CharColor** statement sets the character color of the selection to the specified color, where *Color* is one of the following numeric codes.

Color	**Explanation**
0 (zero)	Auto, specified in the Control Panel (Windows)
1	Black
2	Blue
3	Cyan
4	Green
5	Magenta
6	Red
7	Yellow
8	White
9	Dark Blue
10	Dark Cyan
11	Dark Green
12	Dark Magenta
13	Dark Red
14	Dark Yellow
15	Dark Gray
16	Light Gray

The **CharColor()** function returns the same number codes set by the **CharColor** statement or -1 if all the selected text is not the same color.

See also

Character Formatting Statements and Functions

FormatFont

SelectCurColor

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CopyFormat Example

This example copies the character formatting of the first character of the current paragraph (selected using the predefined bookmark "\Para") to the paragraph immediately above it:

```
EditGoTo "\Para"
CopyFormat
ParaUp 2
ParaDown 1, 1
PasteFormat
```

CopyFormat

¬Example

CopyFormat

Copies the character formatting of the first character of the selected text to another text selection. If a paragraph mark is selected, Word copies paragraph formatting in addition to character formatting. For **CopyFormat** to work, macro instructions must make a selection immediately before the **CopyFormat** instruction, make a new selection immediately following **CopyFormat**, and then use the **PasteFormat** statement to apply the formatting.

_____

See also

Character Formatting Statements and Functions

CopyText

PasteFormat

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountFonts() Example

This example creates a new document and then inserts the name of the current printer along with a list of the fonts available:

```
Dim dlg As FilePrintSetup
GetCurValues dlg
FileNewDefault
Bold 1 : Insert dlg.Printer
InsertPara : Bold 0
For count = 1 To CountFonts()
    Insert Font$(count)
    InsertPara
Next
```

CountFonts()

¬Example

CountFonts()

Returns the number of fonts available with the selected printer. This is the number of fonts listed in the Font dialog box (Format menu) or in the Formatting toolbar's font list.

The font list includes fonts installed on the printer, TrueType fonts (if they are installed), and Windows system fonts.

_____

See also

Character Formatting Statements and Functions

Font

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountLanguages() Example

This example fills the array langnames$() with the list of available language formats:

```
Dim langnames$(CountLanguages())
For count = 1 To CountLanguages()
    langnames$(count) = Language$(count)
Next
```

CountLanguages()

¬Example

CountLanguages()

Returns the number of available language formats, including the No Proofing format (set with the instruction Language "0") and the absence of any language format (set with the instruction Language "(none)"). For the list of valid foreign language names, see ToolsLanguage.

See also

Character Formatting Statements and Functions

Language

ToolsLanguage

DottedUnderline, DottedUnderline()

DottedUnderline [*On*]

DottedUnderline()

The **DottedUnderline** statement adds or removes the dotted-underline character format for the selected text, or controls dotted-underline formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the dotted-underline format: 1 Formats the selection with the dotted-underline format 0 (zero) Removes the dotted-underline format Omitted Toggles the dotted-underline format

The **DottedUnderline()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is in the dotted-underline format
-1	If part of the selection is in the dotted-underline format
1	If all the selection is in the dotted-underline format

See also

Character Formatting Statements and Functions

DoubleUnderline

FormatBordersAndShading

FormatFont

Underline

DoubleUnderline, DoubleUnderline()

DoubleUnderline [*On*]

DoubleUnderline()

The **DoubleUnderline** statement adds or removes the double-underline character format for the selected text, or controls double-underline formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the double-underline format: 1 Formats the selection with the double-underline format 0 (zero) Removes the double-underline format Omitted Toggles the double-underline format

The **DoubleUnderline()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is in the double-underline format
-1	If part of the selection is in the double-underline format
1	If all the selection is in the double-underline format

See also

Character Formatting Statements and Functions

DottedUnderline

FormatBordersAndShading

FormatFont

Underline

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Font Examples

This example selects the paragraph containing the insertion point (using the predefined bookmark "\Para") and then applies 8-point Courier font:

```
EditGoTo "\Para"
Font "Courier", 8
```

The following example displays a message box if the paragraph containing the insertion point has more than one font:

```
EditGoTo "\Para"
If Font$() = "" Then
    MsgBox "Paragraph contains more than one font."
End If
```

Font, Font$()

¬Example

Font *Name$* [**,** *Size*]

Font$()

Font$(*Count***)**

The **Font** statement applies the specified font to the selection.

Argument	Explanation
Name$	The name of the font to apply.
Size	The size of the font, in points. You can use this argument instead of following a **Font** instruction with a **FontSize** instruction.

The **Font$()** function returns the name of the font applied to the selection. If the selection contains more than one font, an empty string ("") is returned. If *Count* is specified, **Font$()** returns the name of the font at position *Count* in the current font list, in the range 1 to **CountFonts()**.

See also

Character Formatting Statements and Functions

CountFonts()

FontSize

FormatFont

FontSize, FontSize()

FontSize *Size*

FontSize()

The **FontSize** statement sets the font size of the selected text, in points.

The **FontSize()** function returns the font size of the selected text. If the selection contains more than one font size, 0 (zero) is returned.

_____

See also

Character Formatting Statements and Functions

Font

FontSizeSelect

FormatFont

FontSizeSelect

FontSizeSelect

If the Formatting toolbar is displayed, moves the focus to the Font Size box; otherwise, displays the Font dialog box (Format menu), selects the Font tab, and then moves the focus to the Size box.

_____

See also

Character Formatting Statements and Functions

Font

FontSize

FormatFont

FontSubstitution

FontSubstitution .UnavailableFont = *text***, .SubstituteFont =** *text*

Sets font mapping options for the active document. The arguments for the **FontSubstitution** statement correspond to the options in the Font Substitution dialog box (Compatibility tab, Options dialog box, Tools menu).

Argument	Explanation
.UnavailableFont	The name of a font not available on your computer that you want to map to a different font for display and printing
.SubstituteFont	The name of a font available on your computer that you want to substitute for the unavailable font

See also

Character Formatting Statements and Functions

ToolsOptionsCompatibility

FormatAddrFonts

FormatAddrFonts [**.Points** = *number*] [**, .Underline** = *number*] [**, .Color** = *number*] [**, .Strikethrough** = *number*] [**, .Superscript** = *number*] [**, .Subscript** = *number*] [**, .Hidden** = *number*] [**, .SmallCaps** = *number*] [**, .AllCaps** = *number*] [**, .Spacing** = *number*] [**, .Position** = *number or text*] [**, .Kerning** = *number*] [**, .KerningMin** = *number or text*] [**, .Default**] [**, .Tab** = *number*] [**, .Font** = *text*] [**, .Bold** = *number*] [**, .Italic** = *number*]

Sets character formatting for the address on an envelope. Precede **ToolsCreateEnvelope** with **FormatAddrFonts** to specify options corresponding to those in the Font dialog box (Format menu). Include .Default to store the options you specify as the default character formatting for the address. For argument descriptions, see FormatFont.

See also

Character Formatting Statements and Functions

FormatFont

FormatRetAddrFonts

ToolsCreateEnvelope

FormatChangeCase

FormatChangeCase [**.Type =** *number*]

Changes the case of the selected text according to the specified type of capitalization. The arguments for the **FormatChangeCase** statement correspond to the options in the Change Case dialog box (Format menu).

Argument	Explanation
.Type	The type of capitalization you want to apply: 0 (zero) or omitted Sentence case (capitalizes the first character in each selected sentence). 1 lowercase (changes selected characters to lowercase). 2 UPPERCASE (changes selected characters to uppercase). 3 Title Case (capitalizes the first letter of each selected word). 4 tOGGLE cASE (toggles the capitalization of each selected letter).

See also

Character Formatting Statements and Functions

AllCaps

ChangeCase

LCase$()

SmallCaps

UCase$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatFont Example

This example demonstrates how measurements other than twips or points may be used in **FormatFont** instructions by specifying a text measurement. This instruction positions the selected characters .25 centimeter above the baseline.

```
FormatFont .Position = ".25 cm"
```

FormatFont

¬Example

FormatFont [**.Points** = *number*] [**, .Underline** = *number*] [**, .Color** = *number*] [**, .Strikethrough** = *number*] [**, .Superscript** = *number*] [**, .Subscript** = *number*] [**, .Hidden** = *number*] [**, .SmallCaps** = *number*] [**, .AllCaps** = *number*] [**, .Spacing** = *number*] [**, .Position** = *number or text*] [**, .Kerning** = *number*] [**, .KerningMin** = *number or text*] [**, .Default**] [**, .Tab** = *number*] [**, .Font** = *text*] [**, .Bold** = *number*] [**, .Italic** = *number*]

Applies character formatting to the selected text. The arguments for the **FormatFont** statement correspond to the options in the Font dialog box (Format menu).

Argument	Explanation
.Points	The font size, in points.
.Underline	Applies a type of underlining: 0 (zero) None 1 Single 2 Words Only 3 Double 4 Dotted
.Color	Color of the text (for a list of colors, see CharColor).
.Strikethrough	If 1, applies strikethrough formatting.
.Superscript	If 1, applies superscript formatting.
.Subscript	If 1, applies subscript formatting.
.Hidden	If 1, applies hidden formatting.
.SmallCaps	If 1, applies small caps formatting.
.AllCaps	If 1, applies all caps formatting.
.Spacing	The spacing between characters, in twips (20 twips = 1 point; 72 points = 1 inch) or a text measurement: 0 (zero) Normal >0 Expanded by the specified distance <0 Condensed by the specified distance
.Position	The character's position relative to the baseline, in units of 0.5 point or a text measurement: 0 (zero) Normal >0 Raised by the specified distance <0 Lowered by the specified distance
.Kerning	If 1, enables automatic kerning (character spacing).
.KerningMin	If .Kerning is set to 1, sets a minimum font size, in points, for automatic kerning.
.Default	Sets the character formats of the

	Normal style.
.Tab	Specifies which tab to select when you display the Font dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) Font tab 1 Character Spacing tab
.Font	The name of the font
.Bold	If 1, applies bold formatting.
.Italic	If 1, applies italic formatting.

See also

Character Formatting Statements and Functions

AllCaps

Bold

CharColor

DoubleUnderline

EditFindFont

EditReplaceFont

Font

FontSize

FormatChangeCase

FormatDefineStyleFont

GrowFont

Hidden

Italic

ResetChar

ShrinkFont

SmallCaps

Strikethrough

Subscript

Superscript

Underline

WordUnderline

FormatRetAddrFonts

FormatRetAddrFonts [**.Points** = *number*] [**, .Underline** = *number*] [**, .Color** = *number*] [**, .Strikethrough** = *number*] [**, .Superscript** = *number*] [**, .Subscript** = *number*] [**, .Hidden** = *number*] [**, .SmallCaps** = *number*] [**, .AllCaps** = *number*] [**, .Spacing** = *number*] [**, .Position** = *number or text*] [**, .Kerning** = *number*] [**, .KerningMin** = *number or text*] [**, .Default**] [**, .Tab** = *number*] [**, .Font** = *text*] [**, .Bold** = *number*] [**, .Italic** = *number*]

Sets character formatting for the return address on an envelope. Precede **ToolsCreateEnvelope** with **FormatRetAddrFonts** to specify options corresponding to those in the Font dialog box (Format menu). Include .Default to store the options you specify as the default character formatting for the return address. For argument descriptions, see FormatFont.

See also

Character Formatting Statements and Functions

FormatAddrFonts

FormatFont

ToolsCreateEnvelope

GrowFont

GrowFont

Increases the font size of the selected text (or text to be inserted at the insertion point) to the next size supported by the selected printer. If more than one font size is included in the selection, each size is increased to the next available setting. If there is no selection, the larger font size will be applied to new text.

_____

See also

Character Formatting Statements and Functions

FontSize

FormatFont

GrowFontOnePoint

ShrinkFont

ShrinkFontOnePoint

GrowFontOnePoint

GrowFontOnePoint

Increases the font size of the selected text (or text to be inserted at the insertion point) by 1 point, whether or not the new size is supported by the selected printer. If more than one font size is included in the selection, each size is increased by 1 point.

_____

See also

Character Formatting Statements and Functions

FontSize

FormatFont

GrowFont

ShrinkFont

ShrinkFontOnePoint

Hidden, Hidden()

Hidden [*On*]

Hidden()

The **Hidden** statement adds or removes the hidden character format for the current selection, or controls hidden formatting for characters to be inserted at the insertion point. You can control the display of hidden text with the Hidden Text check box, an option on the View tab in the Options dialog box (Tools menu).

Argument	Explanation
On	Specifies whether to add or remove the hidden format: 1 Adds the hidden format. 0 (zero) Removes the hidden format. Omitted Toggles the hidden format.

The **Hidden()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as hidden
-1	If part of the selection is formatted as hidden
1	If all the selection is formatted as hidden

See also

Character Formatting Statements and Functions

FormatFont

Italic, Italic()

Italic [*On*]

Italic()

The **Italic** statement adds or removes the italic character format for the selected text, or controls italic formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the italic format: 1 Formats the selection as italic. 0 (zero) Removes the italic format. Omitted Toggles the italic format.

The **Italic()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as italic
-1	If part of the selection is formatted as italic
1	If all the selection is formatted as italic

See also

Character Formatting Statements and Functions

FormatFont

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Language Examples

These instructions show three different uses for the **Language** statement. The first marks the selection as British English, the second as whatever language is twelfth, and the third as No Proofing. The proofing tools skip over text marked as No Proofing.

```
Language "English (UK)"
Language Language$(12)
Language "0"
```

The following macro creates a two-column list in the active document showing all valid **Language$(***Count***)** instructions and the corresponding language names that each value of *Count* returns. The returned language names are all valid arguments for the **Language** statement.

```
Sub MAIN
For i = 1 To CountLanguages()
    If i > 1 Then InsertPara
    Insert "Language$(" + Right$(Str$(i), Len(Str$(i))-1) + ")" \
        + Chr$(9) + Language$(i)
Next
End Sub
```

Language, Language$()

¬Example

Language *Language$*

Language$([*Count*]**)**

The **Language** statement identifies the selected text as being in a specific language. The proofing tools use dictionaries of the specified language on this text. Although the Language dialog box (Tools menu) lists language names in their English form, *Language$* must be in the specified language and include any accented characters. For example, Italian must be specified as "Italiano" and French must be specified as "Français." For a list of valid foreign language names, see ToolsLanguage.

If *Count* is 0 (zero) or omitted, the **Language$()** function returns the language format of the first character of the selected text. If there is no selection, **Language$()** returns the language format of the character to the left of the insertion point.

If *Count* is greater than 0 (zero), **Language$()** returns the name of the language *Count*, where *Count* is the position of the language in an internal list of untranslated language names. Note that the list of untranslated language names does not match the list in the Language dialog box (Tools menu) because in the dialog box, language names are translated into English and sorted alphabetically. The *Count* argument is in the range 1 to **CountLanguages()**.

See also

Character Formatting Statements and Functions

CountLanguages()

ToolsLanguage

NormalFontPosition

NormalFontPosition

Restores the selected characters to the baseline if they have been raised or lowered.

_____

See also

Character Formatting Statements and Functions

FormatFont

NormalFontSpacing

NormalFontSpacing

NormalFontSpacing

Restores the selected characters to Normal character spacing if their current spacing is expanded or condensed.

_____

See also

Character Formatting Statements and Functions

FormatFont

NormalFontPosition

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PasteFormat Example

This example copies both character and paragraph formats from the current paragraph (selected using the predefined bookmark "\Para"), and then applies only the paragraph formats to the next paragraph. Character formats are not applied because no text is selected when PasteFormat runs. **ResetChar** ensures that any new text typed at the insertion point has the font characteristics of the current style.

```
SelType 1                    'Cancel selection, if any
EditGoTo "\Para"
CopyFormat
ParaDown
PasteFormat
ResetChar
```

PasteFormat

¬Example

PasteFormat

Applies formatting copied with **CopyFormat** to the selection. If a paragraph mark was selected when **CopyFormat** was run, Word applies paragraph formatting in addition to character formatting.

_____

See also

Character Formatting Statements and Functions

CopyFormat

ResetChar

ResetPara

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ResetChar Example

If the current paragraph is formatted as Normal, this example selects the entire paragraph (using the predefined bookmark "\Para") and removes any character formatting not defined as part of the style:

```
If StyleName$() = "Normal" Then
    EditGoTo "\Para"
    ResetChar
End If
```

ResetChar, ResetChar()

¬Example

ResetChar

ResetChar()

The **ResetChar** statement removes manual character formatting (formatting not defined in the selected paragraph styles) from the selected text. For example, if you manually format a word or phrase in a paragraph as bold text and the paragraph style is plain text, **ResetChar** would remove the bold format.

The **ResetChar()** function returns the following values without resetting character formats.

Value	Explanation
0 (zero)	If the selected text contains any manual character formatting
1	If the selected text contains no manual character formatting

See also

Character Formatting Statements and Functions

FormatFont

ResetPara

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ShrinkFont Example

This macro inserts a line of increasingly smaller *Z* characters. Note how you can use a colon to place separate instructions on the same line. In this macro, **ShrinkFont** is executed three times in a single line.

```
Sub MAIN
FontSize 45
For count = 1 To 7
    Insert "Z"
    ShrinkFont : ShrinkFont : ShrinkFont
Next count
End Sub
```

ShrinkFont

¬Example

ShrinkFont

Decreases the size of the selected text to the next available font size supported by the assigned printer. If the selection contains characters of varying font sizes, each size is reduced to the next available setting. If there is no selection, the smaller font size will be applied to new text.

See also

Character Formatting Statements and Functions

Font

FontSize

FormatFont

GrowFont

ResetChar

ShrinkFontOnePoint

ShrinkFontOnePoint

ShrinkFontOnePoint

Decreases the font size of the selected text (or text to be inserted at the insertion point) by 1 point, whether or not the new size is supported by the selected printer. If more than one font size is included in the selection, each size is decreased by 1 point.

_____

See also

Character Formatting Statements and Functions

GrowFontOnePoint

ShrinkFont

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SmallCaps Example

If the current paragraph (selected using the predefined bookmark "\Para") has any small caps formatting, this example formats the entire paragraph as small caps:

```
SelType 1                          'Cancel selection, if any
EditGoTo "\Para"
If SmallCaps() = -1 Then SmallCaps 1
```

SmallCaps, SmallCaps()

¬Example

SmallCaps [*On*]

SmallCaps()

The **SmallCaps** statement adds or removes the small caps character format for the current selection, or controls small caps formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the small caps format: 1 Formats the selection as small caps. 0 (zero) Removes the small caps format. Omitted Toggles the small caps format.

The **SmallCaps()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as small caps
-1	If part of the selection is formatted as small caps
1	If all the selection is formatted as small caps

See also

Character Formatting Statements and Functions

AllCaps

ChangeCase

FormatFont

UCase$()

Strikethrough, Strikethrough()

Strikethrough [*On*]

Strikethrough()

The **Strikethrough** statement adds or removes the strikethrough character format for the current selection, or controls strikethrough formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the strikethrough format: 1 Formats the selection as strikethrough. 0 (zero) Removes the strikethrough format. Omitted Toggles the strikethrough format.

The **Strikethrough()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as strikethrough
-1	If part of the selection is formatted as strikethrough
1	If all the selection is formatted as strikethrough

When revision marking is activated, Word uses strikethrough to mark deleted text. However, Word does not recognize text formatted with the **Strikethrough** statement as deleted. If you want to mark the selected text as deleted for the purpose of revision marks, use the following instructions:

```
ToolsRevisions .MarkRevisions = 1
EditClear
```

See also

Character Formatting Statements and Functions

FormatFont

ToolsRevisions

Subscript, Subscript()

Subscript [*On*]

Subscript()

The **Subscript** statement adds or removes the subscript character format for the current selection, or controls subscript formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the subscript format: 1 Formats the selection as subscript. 0 (zero) Removes the subscript format. Omitted Toggles the subscript format.

The **Subscript()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as subscript
-1	If part of the selection is formatted as subscript
1	If all the selection is formatted as subscript

See also

Character Formatting Statements and Functions

FormatFont

Superscript

Superscript, Superscript()

Superscript [*On*]

Superscript()

The **Superscript** statement adds or removes the superscript character format for the current selection, or controls superscript formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the superscript format: 1 Formats the selection as superscript. 0 (zero) Removes the superscript format. Omitted Toggles the superscript format.

The **Superscript()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted as superscript
-1	If part of the selection is formatted as superscript
1	If all the selection is formatted as superscript

See also

Character Formatting Statements and Functions

FormatFont

Subscript

SymbolFont

SymbolFont [*TextToInsert$*]

Formats the selected text with the Symbol font or inserts the specified text formatted with the Symbol font at the insertion point. If no text is specified or selected, the **SymbolFont** statement has no effect.

See also

Character Formatting Statements and Functions

FormatFont

InsertSymbol

Underline, Underline()

Underline [*On*]

Underline()

The **Underline** statement adds or removes the single-underline character format for the selected text, or controls single-underline formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the single-underline format: 1 Formats the selection with the single-underline format. 0 (zero) Removes the single-underline format. Omitted Toggles the single-underline format.

The **Underline()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is in the single-underline format
-1	If part of the selection is in the single-underline format
1	If all the selection is in the single-underline format

See also

Character Formatting Statements and Functions

DottedUnderline

DoubleUnderline

FormatFont

WordUnderline

WordUnderline, WordUnderline()

WordUnderline [*On*]

WordUnderline()

The **WordUnderline** statement adds or removes the word-underline character format for the selected text, or controls word-underline formatting for characters to be inserted at the insertion point.

Argument	Explanation
On	Specifies whether to add or remove the word-underline format: 1 Formats the selection with the word-underline format. 0 (zero) Removes the word-underline format. Omitted Toggles the word-underline format.

The **WordUnderline()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is in the word-underline format
-1	If part of the selection is in the word-underline format
1	If all the selection is in the word-underline format

See also

Character Formatting Statements and Functions

DottedUnderline

DoubleUnderline

FormatFont

Underline

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

AddButton Example

This example replaces the button at position 11 on the Formatting toolbar with button image 79. The built-in command assigned to the button is Color, which requires that an additional value be specified (in this case, "2" for the color blue).

```
DeleteButton "Formatting", 11, 0
AddButton "Formatting", 11, 1, "Color", 79, 0, "2"
```

AddButton

¬Example

AddButton *Toolbar$*, *Position*, *Category*, *Name$* [, *ButtonTxtOrImageNum*[*$*]] [, *Context*] [, *CommandValue$*]

Adds to a toolbar a button that runs a built-in command or macro, applies a font or style, or inserts an AutoText entry.

Argument	Explanation
Toolbar$	The name of the toolbar as listed in the Toolbars dialog box (View menu).
Position	A number corresponding to the position on the specified toolbar at which to add the button, where 1 is the first position, 2 is the second position, and so on. Note that a list box or space counts as one position.
Category	The type of item to be assigned: 1 Built-in commands 2 Macros 3 Fonts 4 AutoText entries 5 Styles
Name$	The name of the built-in command, macro, font, AutoText entry, or style to associate with the button. To add a space to a toolbar, specify an empty string ("").
ButtonTxtOrImageNum[*$*]	The text you want to appear on the button, or a number corresponding to an image for the button in the built-in set of buttons, where 0 (zero) is no image (for a blank button), 1 is the first image, 2 is the second image, and so on. For a list of button images and their associated numbers, see Toolbar Button Images and Numbers.
Context	Determines where the new toolbar assignment is stored: 0 (zero) or omitted Normal template 1 Active template
CommandValue$	Additional text, if any, required for the command specified by *Name$*. For example, if *Name$* is "Color," *CommandValue$* specifies the color. For more information, see the second table in ToolsCustomizeMenus.

See also

Customization Statements and Functions

ChooseButtonImage

CopyButtonImage

DeleteButton

EditButtonImage

MoveButton

PasteButtonImage

ResetButtonImage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ChooseButtonImage Examples

This example changes the image on the Stop button on the Macro toolbar to button image 50:

```
ChooseButtonImage .Face = 50, .Button = 9, .Context = 0, \
        .Toolbar = "Macro"
```

The following example creates a dialog record for the Custom Button dialog box (Toolbar Customization shortcut menu) and then displays the dialog box so you can modify the third button on the Standard toolbar. Note that, unlike those for other dialog records, instructions that set values in a **ChooseButtonImage** dialog record must precede the **GetCurValues** instruction.

```
Dim dlg As ChooseButtonImage
dlg.Toolbar = "Standard"
dlg.Button = 3
GetCurValues dlg
Dialog dlg
ChooseButtonImage dlg
```

ChooseButtonImage

¬Example

ChooseButtonImage [**.Face =** *number***,**] **.Button =** *number***,** [**.Context =** *number***,**] [**.Text** = *text***,**] **.Toolbar =** *text*

Changes the image or text on the specified toolbar button.

Argument	Explanation
.Face	A number corresponding to an image for the button in the built-in set of button images, where 0 (zero) is no image (for a blank button), 1 is the first image, 2 is the second image, and so on. For a list of button images and their associated numbers, see Toolbar Button Images and Numbers.
.Button	A number corresponding to the position of the button to change on the specified toolbar, where 1 is the first position, 2 is the second, and so on. Note that a list box or space counts as one position. If you specify a position that corresponds to a space, Word modifies the button to the right of the space.
.Context	Determines where the toolbar change is stored: 0 (zero) or omitted Normal template 1 Active template
.Text	The text you want to appear on the button. If you specify both .Face and .Text, .Text takes precedence.
.Toolbar	The name of the toolbar as listed in the Toolbars dialog box (View menu).

Note

You can choose a button image for a list box on a toolbar; however, the button is displayed only when the toolbar is vertical.

See also

Customization Statements and Functions

AddButton

CopyButtonImage

EditButtonImage

MoveButton

PasteButtonImage

ResetButtonImage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CopyButtonImage Example

This example inserts the face of the third button on the Standard toolbar (the Save button) onto the first button of a custom toolbar named "TestBar":

```
CopyButtonImage "Standard", 3
PasteButtonImage "TestBar", 1
```

CopyButtonImage

¬Example

CopyButtonImage *Toolbar$*, *Tool* [, *Context*]

Copies the face of the specified toolbar button to the Clipboard.

Argument	Explanation
Toolbar$	The name of the toolbar, as it appears in the Toolbars dialog box (View menu).
Tool	A number corresponding to the button face to copy, where 1 is the first button on the specified toolbar, 2 is the second, and so on.
Context	Specifies which button face Word copies: 0 (zero) or omitted — The button face that is displayed when a document based on the Normal template is active. 1 — The button face that is currently displayed. Note that the button face that is displayed depends on the custom settings, if any, of the active template, any loaded global templates, and the Normal template.

See also

Customization Statements and Functions

AddButton

ChooseButtonImage

EditButtonImage

MoveButton

PasteButtonImage

ResetButtonImage

CountKeys()

CountKeys([*Context*]**)**

Returns the number of key assignments specified on the Keyboard tab in the Customize dialog box (Tools menu) that differ from the default assignments.

Argument	Explanation
Context	Specifies the template in which to count key assignments: 0 (zero) or omitted Normal template 1 Active template

For an example, see KeyCode() Example.

See also

Customization Statements and Functions

KeyCode()

KeyMacro$()

ToolsCustomizeKeyboard

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountMenuItems() Example

This example inserts a tab-delimited list of the menu items on the Tools menu in the active template:

```
tab$ = Chr$(9)
For n = 1 To CountMenuItems("T&ools", 0, 1)
    Insert Str$(n) + tab$ + MenuItemText$("T&ools", 0, n, 1)
    InsertPara
Next
```

CountMenuItems()

¬Example

CountMenuItems(*Menu$***,** *Type* [**,** *Context*]**)**

Returns the number of menu items on the specified menu. Separators are counted as menu items. Note that the list of filenames on the File menu, the list of windows on the Window menu, and the list of proofing tools on the Tools menu are counted as single items.

Argument	Explanation
Menu$	The name of a menu or a shortcut menu. Menu names appear in the Menu list box, which is on the Menus tab in the Customize dialog box (Tools menu). Including an ampersand (&) before the underlined letter in the menu name is optional (for example, you can specify either "File" or "&File"). Do not include the parenthetical phrases "(No Document)" and "(Shortcut)" even though that text appears in the Customize dialog box.
Type	The type of menu: 0 (zero) Menus on the menu bar when a document is open 1 Menus on the menu bar when no document is open 2 Shortcut menus
Context	Specifies which menu items to count: 0 (zero) or omitted The items that are displayed on the menu when a document based on the Normal template is active. 1 The items that are currently displayed on the menu. Note that the items that are displayed on the menu depend on the custom settings, if any, of the active template, any loaded global templates, and the Normal template.

See also

Customization Statements and Functions

MenuItemMacro$()

MenuItemText$()

CountMenus()

CountMenus(*Type* **[,** *Context***])**

Returns the number of menus of the specified type.

Argument	Explanation
Type	The type of menu to count: 0 (zero) Menus on the menu bar when a document is open 1 Menus on the menu bar when no document is open 2 Shortcut menus
Context	Specifies which menus are counted: 0 (zero) The menus that are available when a document based on the Normal template is active 1 or omitted The menus that are currently available Note that the menus that are available depend on the custom settings, if any, of the active template, any loaded global templates, and the Normal template.

For an example, see MenuText$() Example.

See also

Customization Statements and Functions

CountMacros()

CountMenuItems()

MenuText$()

CountToolbarButtons()

CountToolbarButtons(*Toolbar$* [**,** *Context*]**)**

Returns the number of toolbar buttons on the specified toolbar. Note that spaces and list boxes are counted as "buttons."

Argument	Explanation
Toolbar$	The name of the toolbar as it appears in the Toolbars dialog box (View menu)
Context	Specifies which buttons to count: 0 (zero) The buttons that are available when a document based on the Normal template is active 1 or omitted The buttons that are currently available Note that the buttons that are available depend on the custom settings, if any, of the active template, any loaded global templates, and the Normal template.

For an example, see CountToolbars() Example.

See also

Customization Statements and Functions

CountToolbars()

ToolbarButtonMacro$()

ToolbarName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountToolbars() Example

This example creates a new document and then inserts a list of toolbar names followed by the number of buttons on each toolbar.

```
FileNewDefault
For i = 1 To CountToolbars(0)
    name$ = ToolbarName$(i)
    numbuttons = CountToolbarButtons(name$)
    Insert name$ + "," + Str$(numbuttons) + Chr$(13)
Next i
```

CountToolbars()

¬Example

CountToolbars([*Context*]**)**

Returns the number of toolbars listed in the Toolbars dialog box (View menu). Note that not all toolbars are listed in all circumstances. For example, the macro toolbar only appears in the Toolbars dialog box when at least one macro-editing window is open.

Argument	Explanation
Context	Specifies which toolbars to count: 0 (zero) The toolbars that are available when a document based on the Normal template is active 1 or omitted The toolbars that are currently available Note that the toolbars that are available depend on the custom settings, if any, of the active template, any loaded global templates, and the Normal template.

See also

Customization Statements and Functions

CountToolbarButtons()

ToolbarButtonMacro$()

ToolbarName$()

DeleteButton

DeleteButton *Toolbar$*, *Position* [, *Context*]

Removes a button, list box, or space from a toolbar.

Argument	Explanation
Toolbar$	The name of the toolbar as listed in the Toolbars dialog box (View menu).
Position	A number corresponding to the position of the item to remove, where 1 is the first position, 2 is the second, and so on. Note that a list box or space counts as one position.
Context	Determines where the change to the toolbar is stored: 0 (zero) or omitted Normal template 1 Active template

For an example, see AddButton Example.

See also

Customization Statements and Functions

AddButton

ChooseButtonImage

CopyButtonImage

EditButtonImage

MoveButton

PasteButtonImage

ResetButtonImage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditButtonImage Example

This example opens the image editor and displays the first button image on the Standard toolbar for editing:

```
EditButtonImage "Standard", 1, 0
```

EditButtonImage

¬Example

EditButtonImage *Toolbar$*, *Position* [, *Context*]

Runs the button image editor to modify the specified toolbar button image.

Argument	Explanation
Toolbar$	The name of the toolbar as listed in the Toolbars dialog box (View menu).
Position	A number corresponding to the position of the button whose image you want to edit, where 1 is the first position, 2 is the second, and so on. Note that a list box or space counts as one position.
Context	Determines where the new button image is stored: 0 (zero) or omitted Normal template 1 Active template

See also

Customization Statements and Functions

ChooseButtonImage

CopyButtonImage

PasteButtonImage

ResetButtonImage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

KeyCode() Example

This example inserts a table of key codes and macro names for key assignments in the active template that differ from the default assignments:

```
If CountKeys(1) = 0 Then
    MsgBox "No custom key assignments in active template."
    Goto bye
End If
FormatTabs .ClearAll
FormatTabs .Position = "1in", .Set
Bold 1 : Underline 1
Insert "Key Code" + Chr$(9) + "Macro"
Bold 0 : Underline 0
For i = 1 To CountKeys(1)
    Insert Chr$(13) + Str$(KeyCode(i, 1))
    combo2 = KeyCode(i, 1, 2)
    If combo2 <> 255 Then Insert "," + Str$(combo2)
    Insert Chr$(9) + KeyMacro$(i, 1)
Next i
bye:
```

KeyCode()

¬Example

KeyCode(*Count* [**,** *Context*] [**,** *FirstOrSecond*]**)**

Returns a number representing a key assignment that differs from the default assignment. For tables of key codes and the keys they represent, see ToolsCustomizeKeyboard.

Argument	Explanation
Count	A number in the range 1 to **CountKeys()** specifying the custom key assignment for which you want to return a key code
Context	The template containing the custom key assignment: 0 (zero) or omitted Normal template 1 Active template
FirstOrSecond	If the key assignment is a sequence of two key combinations (for example, if you've assigned CTRL+S, N to **NormalStyle**), specifies which key combination to return a code for: 1 The first key combination in the sequence 2 The second key combination in the sequence If you specify 2 and the key assignment is not a key sequence, **KeyCode()** returns 255 (the null key code).

See also

Customization Statements and Functions

CountKeys()

KeyMacro$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MenuItemText$() Example

This example displays a message box with the menu text of the first item on the Help menu when no document is open:

```
MsgBox MenuItemText$("Help", 1, 1, 0)
```

MenuItemText$()

¬Example

MenuItemText$(***Menu$*,** *Type***,** *Item* [**,** *Context*]**)**

Returns the menu text associated with a macro or built-in command assigned to the specified menu. You can change the menu text with **ToolsCustomizeMenus**.

Note

The menu text for a subset of built-in commands changes depending on the current conditions of the Word environment. For these commands, **MenuItemText$()** returns the command name instead of the menu text. The commands include **EditCopy**, **EditPaste**, **EditRedoOrRepeat**, **EditUndo**, **FileClose**, **FileCloseAll**, **FileCloseOrCloseAll**, **FileExit**, **FileQuit**, **FileSave**, **FileSaveAll**, **FileSaveAs**, **FormatFrameOrFramePicture**, **TableDeleteGeneral**, **TableInsertGeneral**, **TableSort**, **TableToOrFromText**, and **ToolsProtectUnprotectDocument**.

Argument	Explanation
Menu$	The name of a menu or shortcut menu. Menu names are listed in the Change What Menu box on the Menus tab in the Customize dialog box (Tools menu). Including an ampersand (&) before the underlined letter in the menu name is optional (for example, you can specify either "File" or "&File"). Do not include the parenthetical phrases "(No Document)" and "(Shortcut)", even though that text appears in the Customize dialog box.
Type	The type of menu: 0 (zero) Menus on the menu bar when a document is open 1 Menus on the menu bar when no document is open 2 Shortcut menus
Item	A number representing the item's position on the menu. The number can range from 1 to the number returned by **CountMenuItems()**. Separators between commands are considered items, and if *Item* represents the position of a separator, **MenuItemText$()** returns the string "(Separator)." Note that lists such as the list of recently used files on the File menu or proofing tools on the Tools menu correspond to a single item. If you specify the position of a list, **MenuItemText$()** returns a parenthetical phrase describing the list --- for example, "(List of Recently Used Files") or

"(List of Proofing Tools)".

Context Specifies the menu assignments for which you want to return the menu text:

0 (zero) or omitted	The assignments that are available when a document based on the Normal template is active
1	The assignments that are currently available

Note that the assignments that are available depend on the custom settings, if any, of the active template, any loaded global templates, and the Normal template.

See also

Customization Statements and Functions

CountMenuItems()

CountMenus()

MenuItemMacro$()

MenuText$()

ToolsCustomizeMenus

MenuMode

MenuMode

Activates the menu bar. **MenuMode** corresponds to pressing the ALT key or the F10 key.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MenuText$() Example

This example defines an array to fill with the names of the shortcut menus available in the Normal template and then fills the array:

```
size = CountMenus(2, 0) - 1
Dim shortcutMenus$(size)
For count = 0 To size
    shortcutMenus$(count) = MenuText$(2, count + 1, 0)
Next count
```

MenuText$()

¬Example

MenuText$(*Type***,** *MenuNumber* [**,** *Context*]**)**

Returns the name of a shortcut menu or a menu on the menu bar. If there is an underlined letter in the menu name, **MenuText$()** includes an ampersand (&) before the letter in the returned text.

Argument	Explanation
Type	The type of menu: 0 (zero) Menus on the menu bar when a document is open 1 Menus on the menu bar when no document is open 2 Shortcut menus
MenuNumber	A number in the range 1 to **CountMenus()** for the specified type. If *Type* is 0 (zero) or 1, *MenuNumber* represents a position on the menu bar from left to right (where 1 corresponds to the File menu). If *Type* is 2, *MenuNumber* represents a position in the series of shortcut menus in the Change What Menu box on the Menus tab in the Customize dialog box (Tools menu): The first shortcut menu in the series is 1, the second is 2, and so on.
Context	Specifies the template that contains the name of menu *MenuNumber* you want to return: 0 (zero) or omitted Normal template 1 Active template

See also

Customization Statements and Functions

CountMenuItems()

CountMenus()

MenuItemMacro$()

MenuItemText$()

ToolsCustomizeMenus

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MoveButton Example

This example copies the Show/Hide ¶ button from the Standard toolbar to the end of the Macro toolbar (assuming these toolbars have not been customized already):

```
MoveButton "Standard", 26, "Macro", 18, 1, 0
```

MoveButton

¬Example

MoveButton *SourceToolbar$*, *SourcePosition*, *TargetToolbar$*, *TargetPosition* [, *Copy*] [, *Context*]

Moves or copies a toolbar button, list box, or space to another toolbar or to another position on the same toolbar.

Argument	Explanation
SourceToolbar$	The name of the toolbar containing the item you want to move or copy, as listed in the Toolbars dialog box (View menu).
SourcePosition	The position of the item you want to move or copy, where 1 is the first position, 2 is the second, and so on. Note that a list box or space counts as one position.
TargetToolbar$	The name of the toolbar to which you want to move or copy the item.
TargetPosition	The position to which you want to move or copy the item.
Copy	If 1, copies instead of moves the toolbar item.
Context	Determines where the change to the toolbar or toolbars is stored: 0 (zero) or omitted Normal template 1 Active template

See also

Customization Statements and Functions

AddButton

ChooseButtonImage

CopyButtonImage

DeleteButton

EditButtonImage

PasteButtonImage

ResetButtonImage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MoveToolbar Example

This example displays the Forms toolbar and moves it to the lower-right corner of the Word window:

```
AppMaximize 1
ViewToolbars .Toolbar = "Forms", .Show
MoveToolbar "Forms", 0, 520, 410
```

MoveToolbar

¬Example

MoveToolbar *Toolbar$*, *Dock*, *HorizPos*, *VertPos*

Moves the specified toolbar. If the specified toolbar is not displayed, an error occurs.

Argument	Explanation
Toolbar$	The name of the toolbar to move, as it is listed in the Toolbars dialog box (View menu).
Dock	Specifies whether to anchor the toolbar at the top, bottom, or either side of the Word window or to set it as a floating toolbar over the document window: 0 (zero) The toolbar floats over the document window. 1 The toolbar is anchored at the top of the Word window. 2 The toolbar is anchored at the left of the Word window. 3 The toolbar is anchored at the right of the Word window. 4 The toolbar is anchored at the bottom of the Word window.
HorizPos, VertPos	If *Dock* is 0 (zero), the horizontal (*HorizPos*) and vertical (*VertPos*) distance from the upper-left corner of the Word window to the upper-left corner of the toolbar, in pixels. If *Dock* is nonzero, you must still specify *HorizPos* and *VertPos*. Word moves the toolbar to the nearest available position in the series of docked toolbars.

See also

Customization Statements and Functions

ToolbarName$()

ToolbarState()

ViewToolbars

NewToolbar

NewToolbar .Name = *text* [**, .Context =** *number*]

Creates a new toolbar to which you can add buttons using the **AddButton** and **MoveButton** statements.

Argument	Explanation
.Name	A name for the new toolbar (spaces are allowed).
.Context	The template in which you want to store the toolbar: 0 (zero) or omitted — Normal template; the toolbar always appears in the Toolbars dialog box (View menu). 1 — Active template; the toolbar only appears in the Toolbars dialog box when the template is active.

See also

Customization Statements and Functions

AddButton

CountToolbars()

MoveButton

ToolbarName$()

ViewToolbars

PasteButtonImage

PasteButtonImage *Toolbar$***,** *Tool* [**,** *Context*]

Sets the face of the specified toolbar button to the graphic on the Clipboard.

Argument	Explanation
Toolbar$	The name of the toolbar, as it appears in the Toolbars dialog box (View menu).
Tool	A number corresponding to the button face to change, where 0 (zero) is the first button on the specified toolbar, 1 is the second, and so on.
Context	Determines where the change is stored: 0 (zero) or omitted Normal template 1 Active template

For an example that uses **PasteButtonImage**, see CopyButtonImage Example.

See also

Customization Statements and Functions

ChooseButtonImage

CopyButtonImage

EditButtonImage

ResetButtonImage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

RenameMenu Example

This example renames the Tools menu "Other Tasks" and makes ALT, O the key sequence that activates the menu:

```
RenameMenu "Tools", "&Other Tasks", 0
```

RenameMenu

¬Example

RenameMenu *Menu$*, *NewName$*, *Type* [, *Context*]

Renames the specified menu. If *Menu$* is not one of the menus of the given *Type*, an error occurs.

Argument	Explanation
Menu$	The name of a menu as it appears on the menu bar. Including an ampersand (&) before the underlined letter in the menu name is optional (for example, you can specify either "File" or "&File").
NewName$	The new menu name. An ampersand (&) before a character in the menu name sets the access key, which will be underlined, for displaying the menu.
Type	The type of menu: 0 (zero) Menus on the menu bar when a document is open. 1 Menus on the menu bar when no document is open. Note that you cannot rename shortcut menus.
Context	Specifies where the new menu name is stored: 0 (zero) or omitted Normal template (the new menu name appears regardless of which template is active). 1 Active template (the new menu name appears only when the template is active).

See also

Customization Statements and Functions

MenuText$()

ToolsCustomizeMenus

ResetButtonImage

ResetButtonImage *Toolbar$*, *Tool* [, *Context*]

Resets the face of the specified toolbar button to the graphic originally associated with the command the button runs.

Argument	Explanation
Toolbar$	The name of the toolbar, as it appears in the Toolbars dialog box (View menu)
Tool	A number corresponding to the button face to reset, where 1 is the first button on the specified toolbar, 2 is the second, and so on
Context	Determines where the change is stored: 0 (zero) or omitted Normal template 1 Active template

See also

Customization Statements and Functions

ChooseButtonImage

CopyButtonImage

EditButtonImage

PasteButtonImage

ViewToolbars

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SizeToolbar Example

This example sizes the Standard toolbar on a 640480 VGA screen such that the buttons are displayed in two horizontal rows.

```
MoveToolbar "Standard", 0, 0, 40
SizeToolbar "Standard", 368
```

SizeToolbar

¬Example

SizeToolbar *Toolbar$, Width*

Sizes a floating toolbar to the available width nearest the specified width. If the specified toolbar is anchored at the top, bottom, or either side of the Word window, **SizeToolbar** has no effect.

Argument	Explanation
Toolbar$	The name of the toolbar as listed in the Toolbars dialog box (View menu)
Width	The width of the toolbar, in pixels

See also

Customization Statements and Functions

MoveToolbar

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolbarButtonMacro$() Example

This example fills an array with the names of commands assigned to the Standard toolbar when a document based on the Normal template is active.

```
size = CountToolbarButtons("Standard", 0) - 1
Dim standardmacro$(size)
For i = 0 To size
    standardmacro$(i) = ToolbarButtonMacro$("Standard", i + 1, 0)
Next i
```

ToolbarButtonMacro$()

¬Example

ToolbarButtonMacro$(***Toolbar$*,** *Position* [**,** *Context*]**)**

Returns the name of the built-in command, macro, font, AutoText entry, or style assigned to the specified toolbar button. If you specify a position that corresponds to a space, **ToolbarButtonMacro$()** returns an empty string ("").

Argument	Explanation
Toolbar$	The name of the toolbar as it appears in the Toolbars dialog box (View menu).
Position	A number corresponding to the position of the button on the specified toolbar, where 1 is the first position, 2 is the second position, and so on. Note that a list box or space counts as one position.
Context	Specifies the button for which to return the name: 0 (zero) The button displayed on the toolbar when a document based on the Normal template is active. 1 or omitted The button currently displayed on the toolbar. Note that the button displayed on the toolbar depends on the custom settings, if any, of the active template, any loaded global templates, and the Normal template.

See also

Customization Statements and Functions

CountToolbarButtons()

CountToolbars()

ToolbarName$()

ToolbarName$()

ToolbarName$(*Toolbar* [**,** *Context*]**)**

Returns the name of the specified toolbar as it appears in the Toolbars dialog box (View menu).

Argument	Explanation
Toolbar	A number in the range 1 to **CountToolbars()**, corresponding to the toolbar whose name you want to return
Context	Specifies the list of toolbars from which to return a name for *Toolbar*: 0 (zero) The list of toolbars available to all documents, including those from loaded global templates 1 or omitted The list of all toolbars currently available, including those available only to documents based on the active template

For an example, see CountToolbars() Example.

See also

Customization Statements and Functions

CountToolbarButtons()

CountToolbars()

ToolbarButtonMacro$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolbarState() Example

This example switches to page layout view if the Drawing toolbar is displayed.

```
If ToolbarState("Drawing") Then ViewPage
```

ToolbarState()

¬Example

ToolbarState(*Toolbar$***)**

Returns -1 if the toolbar specified by *Toolbar$* is displayed and 0 (zero) if it is not.

See also

Customization Statements and Functions

ToolbarName$()

ViewRibbon

ViewRuler

ViewStatusBar

ViewToolbars

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsCustomizeKeyboard Examples

This example assigns CTRL+SHIFT+F to **FileFind**:

```
ToolsCustomizeKeyboard .Category = 1, .Name = "FileFind", \
    .KeyCode = 838, .Add
```

The following example removes the key assignment from **FileFind**:

```
ToolsCustomizeKeyboard .Category = 1, .Name = "FileFind", \
    .KeyCode = 838, .Remove
```

ToolsCustomizeKeyboard

¬Example

ToolsCustomizeKeyboard [**.KeyCode** = *number*] [**, .KeyCode2** = *number*] [**, .Category** = *number*] [**, .Name** = *text*] [**, .Add**] [**, .Remove**] [**, .ResetAll**] [**, .CommandValue** = *text*] [**, .Context** = *number*]

Assigns and reassigns shortcut keys to built-in commands, macros, fonts, AutoText entries, styles, or common symbols. The arguments for the **ToolsCustomizeKeyboard** statement correspond to the options on the Keyboard tab in the Customize dialog box (Tools menu).

Argument	Explanation
.KeyCode	A number representing the key combination, as listed later in this entry.
.KeyCode2	A number representing the second key combination in a key sequence. For example, to assign the key sequence CTRL+S, 1 to the Heading 1 style, you set .KeyCode to 339 (for CTRL+S) and .KeyCode2 to 49 (for 1).
.Category	Specifies the kind of item to assign a shortcut key: 1 or omitted Built-in commands 2 Macros 3 Fonts 4 AutoText entries 5 Styles 6 Common symbols
.Name	The name of the command, macro, or other item. If .Category is set to 6, you can specify a symbol by typing or pasting it into the macro-editing window or using a **Chr$()** instruction such as Chr$(167).
.Add	Adds the key assignment or assignments to the specified item.
.Remove	If the specified key code is currently assigned, removes the assignment such that the associated key combination has no effect. If the specified key code is not assigned, .Remove resets the key code to its default action. Note that specifying .Name is not required when removing or resetting a key assignment.
.ResetAll	Resets all key assignments to their defaults.
.CommandValue	Additional text required for the command specified by .Name, if any. For example, if .Name is set

	to Color, .CommandValue specifies the color. For more information, see the table of settings for .CommandValue in ToolsCustomizeMenus.
.Context	Determines where new key assignments are stored: 0 (zero) or omitted Normal template 1 Active template

The following tables list the values you need to specify the appropriate number for .KeyCode. The first table contains values you add to values in the second table when specifying a key combination (for example, you would specify CTRL+S as the sum of 256 and 83, or 339).

Add this	To combine with this key
256	CTRL
512	SHIFT
1024	ALT

Note that function keys F11 through F16 are equivalent to ALT+F1 through ALT+F6. Therefore, the key code corresponding to F11 has the same effect as a key code corresponding to ALT+F1. Similarly, specifying ALT+F11 --- effectively ALT+ALT+F1 --- is no different from specifying ALT+F1.

This key code	Corresponds to
8	BACKSPACE
9	TAB
12	5 on the numeric keypad when NUM LOCK is off
13	ENTER
19	PAUSE
27	ESC
32	SPACEBAR
33	PAGE UP
34	PAGE DOWN
35	END
36	HOME
45	INS
46	DEL
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
96	0 on the numeric keypad
97	1 on the numeric keypad
98	2 on the numeric keypad
99	3 on the numeric keypad
100	4 on the numeric keypad
101	5 on the numeric keypad
102	6 on the numeric keypad
103	7 on the numeric keypad
104	8 on the numeric keypad
105	9 on the numeric keypad
106	* on the numeric keypad
107	+ on the numeric keypad
109	- on the numeric keypad
110	. on the numeric keypad
111	/ on the numeric keypad
112	F1
113	F2

114	F3
115	F4
116	F5
117	F6
118	F7
119	F8
120	F9
121	F10
122	F11
123	F12
124	F13
125	F14
126	F15
127	F16
145	SCROLL LOCK
186	;
187	=
188	,
189	-
190	.
191	/
192	`
219	[
220	\
221	]
222	'

See also

Customization Statements and Functions

CountKeys()

KeyCode()

KeyMacro$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsCustomizeMenus Example

This example adds the macro TestMacro to the Help menu. The menu item for this macro is Test, which appears immediately after Contents. Because .AddBelow is specified, .Position is not required.

```
ToolsCustomizeMenus .Category = 2, .Name = "TestMacro", \
        .MenuType = 0, .Menu = "Help", .MenuText = "&Test", \
        .AddBelow = "Contents", .Context = 0, .Add
```

ToolsCustomizeMenus

¬Example

ToolsCustomizeMenus [**.MenuType** = *number*] [**, .Position** = *number*] [**, .Category** = *number*] [**, .Name** = *text*] [**, .Menu** = *text*] [**, .AddBelow** = *text*] [**, .MenuText** = *text*] [**, .Rename**] [**, .Add**] [**, .Remove**] [**, .ResetAll**] [**, .CommandValue** = *text*] [**, .Context** = *number*]

Changes menu assignments for built-in commands, macros, fonts, AutoText entries, and styles. The arguments for the **ToolsCustomizeMenus** statement correspond to the options on the Menus tab in the Customize dialog box (Tools menu).

Argument	Explanation
.MenuType	The type of menu you want to modify: 0 or omitted — Menus on the menu bar when a document is open 1 — Menus on the menu bar when no document is open 2 — Shortcut menus
.Position	The position on the menu where you want to add or remove an item: -1 or omitted — Automatically determines an appropriate position for adding the item. -2 — Adds the item to the bottom of the menu. *n* — Adds, removes, or renames the item at the specified position, where 1 is the first position, 2 is the second position, and so on. When adding an item, if *n* is greater than the number of existing menu items plus one, the item is added to the bottom of the menu. Note that separators are considered items.
.Category	The type of item to be assigned: 1 or omitted — Built-in commands 2 — Macros 3 — Fonts 4 — AutoText entries 5 — Styles
.Name	The name of the built-in command, macro, font, AutoText entry, or style whose menu assignment you want to change. When adding or removing a separator, specify "(Separator)."
.Menu	The menu you want to change. Menu names are listed in the Change What Menu list box. Including an ampersand (&) before the underlined letter in the menu name is optional (for

	example, you can specify either "File" or "&File"). Do not include the parenthetical phrases "(No Document)" and "(Shortcut)" even though that text appears in the Customize dialog box.
.AddBelow	The text of or command name associated with the menu item after which you want to add the new item. Note that you do not need to specify this argument if you specify .Position. If you specify both arguments, .AddBelow takes precedence.
.MenuText	The text as it will appear on the menu. Place an ampersand (&) in front of the access key, if any. If you want an ampersand to appear in the menu text, include two ampersands. For example, setting .MenuText to "&Examples && Practices" creates the menu item "Examples & Practices."
.Rename	Renames the menu item specified by .Menu and .Name (and .Position, if specified), with the text specified by .MenuText.
.Add	Adds the item to the menu.
.Remove	Removes the item from the menu.
.ResetAll	Resets all menu assignments to their defaults. Note that this argument resets changes made with **ToolsCustomizeMenuBar** as well as **ToolsCustomizeMenus**.
.CommandValue	Additional text required for the command specified by .Name, if any. For example, if .Name is set to Color, .CommandValue specifies the color. For more information, see the next table in this entry.
.Context	Determines where the new menu assignment is stored: 0 (zero) or omitted Normal template 1 Active template

When removing or renaming menu items, you must specify .Name and .Menu. Word starts at the bottom of the specified menu and moves up until it finds an item that performs the action specified by .Name. You can also specify .Position and .MenuText if you need to be sure that the menu item you are changing is at a specific position and has specific menu text. Note that if Word can't find an item that matches each of the specified criteria, an error

occurs.

The following table lists in the left column commands that require a command value and describes in the right column how to specify .CommandValue. The equivalent action in the Customize dialog box (Tools menu) to specifying .CommandValue is selecting an item in the list box that appears when you select one of the following commands in the Commands box.

If .Name is set to	.CommandValue must be
Borders, Color, or Shading	A number --- specified as text --- corresponding to the position of the setting in the list box containing values, where 0 (zero) is the first item, 1 is the second item, and so on.
Columns	A number between 1 and 45 --- specified as text --- corresponding to the number of columns you want to apply.
Condensed	A text measurement between 0.1 pt and 12.75 pt specified in 0.05-point increments (72 points = 1 inch).
Expanded	A text measurement between 0.1 pt and 12.75 pt specified in 0.05-point increments (72 points = 1 inch).
FileOpenFile	The path and filename of the file to open. If the path is not specified, the current directory is assumed.
Font Size	A positive text measurement, specified in 0.5-point increments (72 points = 1 inch).
Lowered, Raised	A text measurement between 1 pt and 64 pt specified in 0.5-point increments (72 points = 1 inch).
Symbol	A string created by concatenating a **Chr$()** instruction and the name of a symbol font (for example, Chr$(167) + "Symbol").

See also

Customization Statements and Functions

CountMenuItems()

MenuItemMacro$()

MenuText$()

ToolsCustomizeMenuBar

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsCustomizeMenuBar Example

This example adds a new menu named Macros to the right of the Help menu:

```
ToolsCustomizeMenuBar .MenuText = "&Macros", .Add, .Position = -1
```

ToolsCustomizeMenuBar

¬Example

ToolsCustomizeMenuBar [**.Context** = *number*] [**, .Position** = *number*] [**, .MenuType** = *number*] [**, .MenuText** = *text*] [**, .Menu** = *text*] [**, .Add**] [**, .Remove**] [**, .Rename**]

Adds, removes, or renames menus on the menu bar. Arguments for the **ToolsCustomizeMenuBar** statement correspond to options in the Menu Bar dialog box (Menus tab, Customize command, Tools menu).

Argument	Explanation
.Context	Determines where the menu change is stored: 0 (zero) or omitted Normal template 1 Active template
.Position	Specifies where to add a new menu: 0 (zero) corresponds to the first (leftmost) position, 1 to the second, and so on. To add a menu to the end of the menu bar, set .Position to -1.
.MenuType	Specifies which menu bar to change: 0 or omitted The menu bar when a document is open 1 The menu bar when no document is open
.MenuText	A new name for the menu; used with .Add and .Rename. Place an ampersand (&) in front of the access key, if any. Note that a menu bar can have only one menu of a given name (irrespective of ampersands).
.Menu	The name of the menu you want to change; used with .Rename and .Remove. Including an ampersand (&) before the underlined letter in the menu name is optional (for example, you can specify either "File" or "&File").
.Add	Adds the menu specified by .MenuText to the menu bar.
.Remove	Removes the specified menu.
.Rename	Renames the specified menu.

If you do not specify .Add, .Remove, or .Rename, Word adds the menu.

To reset menu bars, use the .ResetAll argument with **ToolsCustomizeMenus**. Note that this resets all menu assignments to their defaults.

See also

Customization Statements and Functions

ToolsCustomizeMenus

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Date$() Example

This example displays the current date in a message box, in the form MMMM d, yyyy (which produces a date string such as January 1, 1994). The first instruction saves the original date format so that it may be restored in the last instruction.

```
OriginalFormat$ = GetPrivateProfileString$("Microsoft Word", \
    "DateFormat", "WINWORD6.INI")
SetPrivateProfileString "Microsoft Word", "DateFormat", \
    "MMMM d, yyyy", "WINWORD6.INI"
MsgBox "Today is " + Date$() + "."
SetPrivateProfileString "Microsoft Word", "DateFormat", \
    OriginalFormat$, "WINWORD6.INI"
```

Date$()

¬Example

Date$([*SerialNumber*]**)**

Returns a date corresponding to the *SerialNumber*, a decimal representation of the date, time, or both. If *SerialNumber* is omitted, **Date$()** returns today's date. For information about serial numbers, see DateSerial().

The date format is determined by the "DateFormat=" line in the [Microsoft Word] section of your WINWORD6.INI file. (If there is no "DateFormat=" line, **Date$()** uses the sShortDate setting in the [intl] section of WIN.INI.) Within a macro, you can use **SetPrivateProfileString** to change the current date format.

_____

See also

Date and Time Statements and Functions

DateSerial()

DateValue()

Day()

GetPrivateProfileString$()

Month()

Now()

SetPrivateProfileString

Time$()

Today()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DateSerial() Example

This example prompts the user to enter a date and then displays a message box to indicate the day of the week corresponding to that date:

```
Dim days$(7)
days$(1) = "Sunday" : days$(2) = "Monday" : days$(3) = "Tuesday"
days$(4) = "Wednesday" : days$(5) = "Thursday"
days$(6) = "Friday" : days$(7) = "Saturday"
datestring$ = InputBox$("Please enter the date for which " + \
                                                    "you want to determine the weekday (mm/dd/yy):")
monthnum = Val(Left$(datestring$, 2))
daynum = Val(Mid$(datestring$, 4, 2))
yearnum = Val(Right$(datestring$, 2))
dayofweek = DateSerial(yearnum, monthnum, daynum)
MsgBox "The day of the week is " + days$(Weekday(dayofweek)) + "."
```

DateSerial()

¬Example

DateSerial(*Year***,** *Month***,** *Day***)**

Returns the serial number of the specified date. The serial number corresponds to the number of days between December 30, 1899, and the specified date, up to December 31, 4095. For example, the serial number 1 corresponds to December 31, 1899. **DateSerial()** generates an error if the specified date is beyond the allowable range.

Argument	Explanation
Year	A number between 0 (zero) and 4095, inclusive, or a numeric expression. To specify a year in the range 1900 to 1999, you can give the last two digits of the year; to specify the year 1899 or a year after 1999, give all four digits of the year.
Month	A number between 1 and 12, inclusive, or a numeric expression representing the month of the year.
Day	A number between 1 and 31, inclusive, or a numeric expression representing the day of the month.

See also

Date and Time Statements and Functions

Date$()

DateValue()

Day()

Month()

Now()

TimeSerial()

Today()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DateValue() Example

This example prompts the user to enter an end date, and then displays the number of days between today and that date:

```
enddate$ = InputBox$("Please enter an end date:")
serialenddate = DateValue(enddate$)
numdays = serialenddate - Today()
numdays = Abs(numdays)
MsgBox "The number of days between now and " + enddate$ + \
    " is" + Str$(numdays) + "."
```

DateValue()

¬Example

DateValue(*DateText$***)**

Returns the serial number of the date represented by *DateText$*. Use **DateValue()** to convert a date represented by text to a serial number. A serial number is a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

Argument	Explanation
DateText$	A string representing a date in a Word date format. For example, the following are each valid representations of July 8, 1991: 7/8/91 July 8, 1991 8 July 1991 If *DateText$* includes only numbers, **DateValue()** recognizes the order of the components of the date as month, day, year.

DateText$ must represent a date from December 30, 1899, to December 31, 4095. **DateValue()** generates an error if *DateText$* is out of this range. If the year portion of *DateText$* is omitted, **DateValue()** uses the current year from your computer's built-in calendar. **DateValue()** ignores time information in *DateText$*.

See also

Date and Time Statements and Functions

Date$()

DateSerial()

Now()

TimeValue()

Today()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Day() Example

This example uses the **Day()** function to determine the day of the month, and then displays the result in a message box with the appropriate suffix (for example, "3rd"):

```
daynumber = Day(Now())
Select Case daynumber
    Case 1, 21, 31
        daysuffix$ = "st"
    Case 2, 22
        daysuffix$ = "nd"
    Case 3, 23
        daysuffix$ = "rd"
    Case Else
        daysuffix$ = "th"
End Select
MsgBox "Today is the" + Str$(daynumber) + \
        daysuffix$ + " day of the month."
```

Day()

¬Example

Day(*SerialNumber***)**

Returns an integer between 1 and 31, inclusive, corresponding to the day component of *SerialNumber*, a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

Argument	Explanation
SerialNumber	The serial number used by Word for date and time calculations. *SerialNumber* can represent a date or time (or both) from December 30, 1899, through December 31, 4095, where December 30, 1899, is 0 (zero).

See also

Date and Time Statements and Functions

DateSerial()

Hour()

Minute()

Month()

Now()

Second()

Today()

Weekday()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Days360() Example

This example uses **Days360()** to determine the number of days between January 1, 1992, and February 1, 1993, assuming a 360-day year:

```
numdays = Days360("1/1/92","2/1/93")
```

Days360()

¬Example

Days360(*StartDate*[*$*]**,** *EndDate*[*$*]**)**

Returns the number of days between two dates based on a 360-day year (twelve 30-day months). Use this function to help compute payments if your accounting system is based on twelve 30-day months. The arguments *StartDate* and *EndDate* are the two dates between which you want to know the number of days. If *StartDate* occurs after *EndDate*, **Days360()** returns a negative number.

Argument	Explanation
StartDate[*$*]	A text string or a serial number that represents the initial date. For information about available date formats in Word, see DateValue(). For information about serial numbers, see DateSerial().
EndDate[*$*]	A text string or a serial number that represents the end date.

See also

Date and Time Statements and Functions

DateSerial()

DateValue()

Day()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Hour() Example

This example sets the variable thishour to the hours component of the current time:

```
thishour = Hour(Now())
```

Hour()

¬Example

Hour(*SerialNumber***)**

Returns an integer between 0 (zero) and 23, inclusive, corresponding to the hours component of *SerialNumber*, a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

See also

Date and Time Statements and Functions

DateSerial()

Day()

Minute()

Month()

Now()

Second()

Today()

Weekday()

Year()

InsertDateField

InsertDateField

Inserts a DATE field at the insertion point. The date format matches the last format chosen using the Date And Time command (Insert Menu).

See also

Date and Time Statements and Functions

InsertDateTime

InsertField

InsertPageField

InsertTimeField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertDateTime Example

This example inserts a TIME field for the current date in the form "3 September, 1993":

```
InsertDateTime .InsertAsField = 1, .DateTimePic = "d MMMM, yyyy"
```

InsertDateTime

¬Example

InsertDateTime [**.DateTimePic =** *text*] [**, .InsertAsField =** *number*]

Inserts the current date, time, or both, as either text or a TIME field.

Argument	Explanation
.DateTimePic	A string describing the format used for displaying the date, time, or both. If omitted, Word uses the format specified by the "DateFormat=" line in the [Microsoft Word] section of WINWORD6.INI. If there is no "DateFormat=" line, Word uses the "sShortDate setting" in the [intl] section of WIN.INI.
.InsertAsField	Specifies whether Word inserts the information as a TIME field: 0 (zero) Word inserts the information as text. 1 Word inserts the information as a TIME field. Omitted Word inserts the information according to the current setting of the Insert As Field check box in the Date And Time dialog box (Insert menu).

See also

Date and Time Statements and Functions

InsertDateField

InsertField

InsertPageField

InsertTimeField

InsertTimeField

Inserts a TIME field without any field instructions at the insertion point.

_____

See also

Date and Time Statements and Functions

InsertDateField

InsertDateTime

InsertField

InsertPageField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Minute() Example

This example displays the minutes component of the current time:

```
mins = Minute(Now())
MsgBox "It is" + Str$(mins) + " minute(s) after the hour."
```

Minute()

¬Example

Minute(*SerialNumber***)**

Returns an integer between 0 (zero) and 59, inclusive, corresponding to the minutes component of *SerialNumber,* a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

See also

Date and Time Statements and Functions

DateSerial()

Day()

Hour()

Month()

Now()

Second()

Today()

Weekday()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Month() Example

This example displays the number of full months between the current month and the new year:

```
months = Month(Now())
MsgBox Str$(12 - months) + " full month(s) left in the year."
```

Month()

¬Example

Month(*SerialNumber***)**

Returns an integer between 1 and 12, inclusive, corresponding to the month component of *SerialNumber*, a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

See also

Date and Time Statements and Functions

DateSerial()

Day()

Hour()

Minute()

Now()

Second()

Today()

Weekday()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Now() Example

This example displays a message box stating the number of days until the new year. The number of days is calculated by subtracting the serial number for the current date and time from the serial number for January 1 of the following year.

```
yr = Year(Now())
rightNow = Now()
jan1 = DateSerial(yr + 1, 1, 1)
MsgBox "Days until the new year:" + Str$(jan1 - rightNow)
```

Fractions of a day appear in the result after the decimal point. To return strictly the number of days with no decimal fraction, substitute **Today()** in place of **Now()** in the preceding example.

Now()

¬Example

Now()

Returns a serial number that represents the current date and time according to the computer's system date and time. Numbers to the left of the decimal point represent the number of days between December 30, 1899 and the current date; numbers to the right represent the time as a fraction of a day. For more information about serial numbers, see DateSerial().

_____

See also

Date and Time Statements and Functions

Date$()

DateSerial()

DateValue()

Today()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OnTime Example

This example sets up a simple alarm clock function in Word. The first macro in the example sets up the background timer.

```
'Alarm program: Prompts user to input time for alarm to sound
'Current time appears in title bar of input box
Sub MAIN
Alarm$ = InputBox$("Time? (HH:mm:ss), 24hr", "Alarm " + Time$())
'Set background timer to run macro called Beeper
'No tolerance is set, so alarm will always sound
OnTime Alarm$, "Beeper"
End Sub
```

The following macro sounds an alarm and displays an alert message in response to the timer set up by the preceding macro. This macro must be named "Beeper" to work properly with the preceding macro.

```
Sub MAIN                                    'Beeper program
For count = 1 To 7
    Beep
    Beep
    For TL = 1 To 100                       'Adds a delay between beeps
    Next
Next
MsgBox "Preset alarm sounded", "Beeper " + Time$(), 48
End Sub
```

OnTime

¬Example

OnTime *When*[*$*]**,** *Name$* [**,** *Tolerance*]

Sets up a background timer that runs a macro at the time specified by *When*[*$*]. If Word is occupied at the specified time --- for example, if a dialog box is displayed or Word is performing a large sort operation --- the macro runs as soon as Word is idle.

If you close Word before *When*[*$*], the timer is canceled and is not reactivated when you start Word. To reset the timer each time you start Word---for example, if you want to automatically set an alarm macro at your next Word session (see the example in this entry) --- name the macro that includes the **OnTime** instruction "AutoExec." For information about AutoExec and other auto macros, see Auto Macros.

Argument	Explanation
When[*$*]	The time the macro is to be run, expressed as text in a 24-hour format (*hours***:***minutes***:***seconds*) or as a serial number, a decimal representation of the date, time, or both. For more information on serial numbers, see DateSerial().
	When you specify the time as text, including *seconds* is optional. For example, 2:37 P.M. is expressed as "14:37" and 2:37 A.M. is expressed as "02:37" or "2:37." Midnight is "00:00:00."
	You can include a string representing the date before the time. There are several available date formats, depending on the settings in the [intl] section of WIN.INI. You can control the default date format by choosing the International option in Control Panel (Windows). If the date is not specified, the macro runs at the first occurrence of the specified time.
Name$	The name of the macro to be run. For the macro to run, it must be available both when the **OnTime** instruction is run and when the specified time has arrived. For this reason, it is best to specify a macro in the global template, NORMAL.DOT. The format *MacroName***.***SubroutineName* is not allowed.
Tolerance	If *Name$* has not yet started within *Tolerance* seconds after *When*, Word will not run the macro. If *Tolerance* is 0 (zero) or omitted, Word always runs the

	macro, regardless of how many seconds elapse before Word is idle and can run the macro.

Note

Word can run only one **OnTime** macro at a time. If you start a second, the first **OnTime** macro is canceled.

See also

Date and Time Statements and Functions

Date$()

DateSerial()

DateValue()

Day()

Month()

Now()

TimeValue()

Today()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Second() Example

This example produces a beep five seconds after the macro is started. Note that Second(Now()) returns the seconds component of the current time according to your computer's built-in clock.

```
n = Second(Now())
beepTime = n + 5
If beepTime > 59 Then beepTime = beepTime - 60
While Second(Now()) <> beepTime
Wend
Beep
```

Second()

¬Example

Second(*SerialNumber***)**

Returns an integer between 0 (zero) and 59, inclusive, corresponding to the seconds component of *SerialNumber,* a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

See also

Date and Time Statements and Functions

DateSerial()

Day()

Hour()

Minute()

Month()

Now()

Today()

Weekday()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Time$() Example

When included at the end of a macro, this example displays a message box showing the time the macro finished running:

```
MsgBox "The macro finished at " + Time$()
```

Time$()

¬Example

Time$([*SerialNumber*]**)**

Returns a time corresponding to *SerialNumber,* a decimal representation of the date, time, or both. If *SerialNumber* is omitted, **Time$()** returns the current time. For information about serial numbers, see DateSerial().

The time format is determined by the "TimeFormat=" line in the [Microsoft Word] section of your WINWORD6.INI file. (If there is no "TimeFormat=" line, **Time$()** uses the time settings in the [intl] section of WIN.INI.) Within a macro, you can use **SetPrivateProfileString** to change the current time format.

_____

See also

Date and Time Statements and Functions

Date$()

DateSerial()

GetPrivateProfileString$()

Hour()

Minute()

Now()

Second()

SetPrivateProfileString

TimeSerial()

TimeValue()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TimeSerial() Example

This example displays a message box with a decimal number corresponding to the fraction of the current day that has passed:

```
h = Hour(Now())
m = Minute(Now())
s = Second(Now())
fraction = TimeSerial(h, m, s)
MsgBox Left$(Str$(fraction), 5) + " of this day is over."
```

TimeSerial()

¬Example

TimeSerial(*Hour***,** *Minute***,** *Second***)**

Returns the serial number of the specified time. Because serial numbers are decimal numbers representing a number of days, **TimeSerial()** returns a decimal number from 0 to 0.99998842592593. For more information about serial numbers, see DateSerial().

Argument	Explanation
Hour	A number between 0 and 23, inclusive, or a numeric expression
Minute	A number between 0 and 59, inclusive, representing the minutes after the hour, or a numeric expression
Second	A number between 0 and 59, inclusive, representing the seconds after the minute, or a numeric expression

See also

Date and Time Statements and Functions

DateSerial()

Day()

Month()

Now()

Time$()

TimeValue()

Today()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TimeValue() Example

This example displays the serial number for the current time in the status bar:

```
Print TimeValue(Time$())
```

TimeValue()

¬Example

TimeValue(*TimeText$***)**

Returns the serial number of the time represented by *TimeText$*. Use **TimeValue()** to convert a time represented by text to a serial number. A serial number is a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

Argument	Explanation
TimeText$	A string representing the time. For example, the following are each valid representations of 4:30 P.M.: 16:30:00 4:30 pm 4:30 PM *TimeText$* must represent a time between 00:00:00 and 23:59:59 on the 24-hour clock. **TimeValue()** generates an error if *TimeText$* is out of this range.

See also

Date and Time Statements and Functions

DateSerial()

DateValue()

Day()

Month()

Now()

Time$()

TimeSerial()

Today()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Today() Example

This example displays a message box stating the number of days until the new year. The number of days is calculated by subtracting the serial number for the current date from the serial number for January 1 of the following year.

```
yr = Year(Today())
rightNow = Today()
jan1 = DateSerial(yr + 1, 1, 1)
MsgBox "Days until the new year:" + Str$(jan1-rightNow)
```

Today()

¬Example

Today()

Returns a serial number that represents the current date according to the computer's system date. Unlike **Now()**, which returns a serial number in which the digits after the decimal point represent the time as a fraction of a day, **Today()** returns strictly whole numbers. For more information about serial numbers, see DateSerial().

See also

Date and Time Statements and Functions

Date$()

DateSerial()

DateValue()

Day()

Month()

Now()

Time$()

TimeSerial()

TimeValue()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Weekday() Example

This example defines an array containing the names of the days of the week, returns a number corresponding to the current weekday, and then uses this number with the array to return the name of the current weekday. This name is then displayed in a message box.

```
Dim days$(7)
days$(1) = "Sunday" : days$(2) = "Monday" : days$(3) = "Tuesday"
days$(4) = "Wednesday" : days$(5) = "Thursday"
days$(6) = "Friday" : days$(7) = "Saturday"
thisday = Weekday(Now())
MsgBox "Would you believe it's " + days$(thisday) + " already?"
```

Weekday()

¬Example

Weekday(*SerialNumber***)**

Returns an integer between 1 and 7, inclusive, corresponding to the day of the week (where 1 is Sunday) on which the date represented by *SerialNumber* falls. A serial number is a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

See also

Date and Time Statements and Functions

DateSerial()

Day()

Hour()

Minute()

Month()

Now()

Second()

Today()

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Year() Example

This example returns the year component of the current date, converts it to a string, and then shortens the string to only the final two digits:

```
years = Year(Now())
years$ = Str$(years)
years$ = Right$(years$, 2)
```

Year()

¬Example

Year(*SerialNumber***)**

Returns an integer between 1899 and 4095, inclusive, corresponding to the year component of *SerialNumber*, a decimal representation of the date, time, or both. For information about serial numbers, see DateSerial().

See also

Date and Time Statements and Functions

DateSerial()

Day()

Hour()

Minute()

Month()

Now()

Second()

Today()

Weekday()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DDEExecute Example

This example starts Microsoft Excel if it is not running and then opens a channel to Microsoft Excel and the System topic (for information on the System topic, see DDERequest$()). The example then sends commands to open BUDGET.XLS and go to cell R20C5. Unlike a document, which may or may not be open, the System topic is always available when Microsoft Excel is running.

```
If AppIsRunning("Microsoft Excel") = 0 Then Shell "C:\EXCEL\EXCEL.EXE"
channel = DDEInitiate("Excel", "System")
q$ = Chr$(34)
cmd1$ = "[OPEN(" + q$ + "C:\EXCEL\EXAMPLES\BUDGET.XLS" + q$ + ")]"
cmd2$ = "[SELECT(" + q$ + "R20C5" + q$ + ", " + q$ + "R20C5" + q$ + ")]"
bothcmds$ = cmd1$ + cmd2$
DDEExecute channel, bothcmds$
```

DDEExecute

¬Example

DDEExecute *ChanNum*, *Command$*

Sends a command or series of commands to an application through a dynamic-data exchange (DDE) channel.

Argument	Explanation
ChanNum	The channel number of the DDE conversation as returned by **DDEInitiate()**. If *ChanNum* doesn't correspond to an open channel, an error occurs.
Command$	A command or series of commands recognized by the server application. You can also use the format described under **SendKeys** to send specific key sequences. If the server application can't perform the specified command, an error occurs.

In Microsoft Excel and many other applications that support DDE, *Command$* should be one or more statements or functions in the application's macro language. For example, in Microsoft Excel the XLM macro instruction to create a new worksheet is NEW(1). To send the same command through a DDE channel, you use the following instruction:

```
DDEExecute channel, "[NEW(1)]"
```

Note that some applications, including Microsoft Excel, require that each command received through a DDE channel be enclosed in brackets.

You can use a single **DDEExecute** instruction to send more than one command. For example, the following instruction tells Microsoft Excel to open and then close a worksheet:

```
DDEExecute channel, "[NEW(1)][FILE.CLOSE(0)]"
```

Note that there is no space between the bracketed commands; a space character between the commands would cause an error. The preceding instruction is equivalent to the following two instructions:

```
DDEExecute channel, "[NEW(1)]"
DDEExecute channel, "[FILE.CLOSE(0)]"
```

Many commands require arguments in the form of strings enclosed in quotation marks. Because quotation marks indicate the beginning and end of a string in WordBasic, you must use Chr$(34) to include a quotation mark in a command string. For example, the following instruction tells Microsoft Excel to open BUDGET.XLS:

```
DDEExecute channel, "[OPEN(" + Chr$(34) + "BUDGET.XLS" + Chr$(34) + ")]"
```

See also

DDE Statements and Functions

DDEInitiate()

DDEPoke

DDERequest$()

DDETerminate
DDETerminateAll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DDEInitiate() Examples

This example opens a channel to Microsoft Excel and the file LOAN.XLS. The variable channel is assigned the channel number that is returned. If Microsoft Excel is not running or LOAN.XLS is not open, the function returns 0 (zero) and generates an error.

```
channel = DDEInitiate("Excel", "C:\EXCEL\LOAN.XLS")
```

The following example first ensures that Microsoft Excel is running, and then opens a channel to Microsoft Excel and the System topic and uses **DDERequest$()** to get the Topics item. The Topics item, a standard item in the System topic that is always available, is a list of available topics, including the names of open documents.

```
wordWin$ = "Microsoft Word - " + WindowName$()
If AppIsRunning("Microsoft Excel") = 0 Then
    Shell "C:\EXCEL\EXCEL.EXE"
    AppActivate wordWin$, 1
End If
channel = DDEInitiate("Excel", "System")
topics$ = DDERequest$(channel, "Topics")
If InStr(topics$, "Sheet1") <> 0 Then
    MsgBox "Sheet1 is an available topic."
End If
DDETerminate channel
```

DDEInitiate()

¬Example

DDEInitiate(*Application$***,** *Topic$***)**

Initiates a dynamic-data exchange (DDE) conversation with an application and opens a DDE channel through which the conversation takes place. If **DDEInitiate()** is able to open a channel, it returns the number of the open channel, which is an integer greater than 0 (zero). (The first DDE channel Word opens during a session is channel 1, the second is channel 2, and so on.) All subsequent DDE instructions during the DDE conversation use this number to specify the channel. **DDEInitiate()** returns 0 (zero) if it fails to open a channel.

Argument	Explanation
Application$	The name used to specify an application that supports DDE as a DDE server. This is usually the name of the application's .EXE file without the .EXE filename extension. If the application is not running, **DDEInitiate()** cannot open a channel and returns an error.
Topic$	The name of a topic recognized by *Application$*. An open document is a typical topic. (If *Topic$* is a document name, the document must be open.) If *Application$* doesn't recognize *Topic$*, **DDEInitiate()** generates an error. Many applications that support DDE recognize a topic named System, which is always available and can be used to find out which other topics are available. For more information on the System topic, see DDERequest$().

The maximum number of channels that can be open simultaneously is determined by Microsoft Windows and your system's memory and resources. If you aren't using an open channel, you should conserve resources by closing it using **DDETerminate** or **DDETerminateAll**.

See also

DDE Statements and Functions

DDEExecute

DDEPoke

DDERequest$()

DDETerminate

DDETerminateAll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DDEPoke Examples

This example sends the value of the variable Total$ to the second row and third column of SHEET1.XLS in Microsoft Excel:

```
channel = DDEInitiate ("Excel", "Sheet1")
DDEPoke channel, "R2C3", Total$
```

The following example sends the string "Total: $1,434" to the cell named "QuarterTotal" in SHEET1.XLS:

```
channel = DDEInitiate("Excel", "Sheet1")
DDEPoke channel, "QuarterTotal", "Total: $1,434"
```

DDEPoke

¬Example

DDEPoke *ChanNum*, *Item$*, *Data$*

Uses an open dynamic-data exchange (DDE) channel to send data to an application. When you start a DDE conversation using **DDEInitiate()**, you open a channel to a specific topic recognized by the server application. In Microsoft Excel, for example, each open document is a separate topic. When you send information to a topic in the server application, you must specify the item in that topic you want to send information to. In Microsoft Excel, for example, cells are valid items and they are referred to using either the "R1C1" format or named references.

DDEPoke sends data as a text string; you cannot send text in any other format, nor can you send graphics.

Argument	Explanation
ChanNum	The channel number of the DDE conversation as returned by **DDEInitiate()**. If *ChanNum* doesn't correspond to an open channel, an error occurs.
Item$	An item within a DDE topic. If the server application doesn't recognize *Item$*, an error occurs.
Data$	The string containing the data to send to the server application.

See also

DDE Statements and Functions

DDEExecute

DDEInitiate()

DDERequest$()

DDETerminate

DDETerminateAll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DDERequest$() Example

This example opens a channel to Microsoft Excel and BUDGET.XLS and then requests the contents of cell R5C4:

```
channel = DDEInitiate("Excel", "BUDGET.XLS")
a$ = DDERequest$(channel, "R5C4")
MsgBox a$
```

DDERequest$()

¬Example

DDERequest$(*ChanNum***,** *Item$***)**

Uses an open dynamic-data exchange (DDE) channel to request an item of information from an application. When you start a DDE conversation using **DDEInitiate()**, you open a channel to a specific topic recognized by the server application. In Microsoft Excel, for example, each open document is a separate topic. When you request information from the topic in the server application, you must specify the item in that topic whose contents you are requesting. In Microsoft Excel, for example, cells are valid items and they are referred to using either the "R1C1" format or named references.

DDERequest$() returns data as a text string only; if the function is unsuccessful, it returns an empty string (""). Text in any other format cannot be transferred, nor can graphics.

Argument	Explanation
ChanNum	The channel number of the DDE conversation as returned by **DDEInitiate()**. If *ChanNum* doesn't correspond to an open channel, an error occurs.
Item$	The item within a DDE topic recognized by the server application. **DDERequest$()** returns the entire contents of the specified item. If the server application doesn't recognize *Item$*, an error occurs.

Microsoft Excel and other applications that support DDE recognize a topic named System. Three standard items in the System topic are described in the following table. Note that you can get a list of the other items in the System topic using the item SysItems.

Item in System topic	Effect
SysItems	Returns a list of all items in the System topic
Topics	Returns a list of available topics
Formats	Returns a list of all the Clipboard formats supported by Word

For an example that uses the System topic, see DDETerminate() Example.

See also

DDE Statements and Functions

DDEExecute

DDEInitiate()

DDEPoke

DDETerminate

DDETerminateAll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DDETerminate Example

This example opens a DDE channel to request a list of topics from Microsoft Excel. The instructions insert the requested list into the active Word document and then close the channel.

```
channel = DDEInitiate("Excel", "System")
a$ = DDERequest$(channel, "Topics")
Insert a$
DDETerminate channel
```

DDETerminate

¬Example

DDETerminate *ChanNum*

Closes the specified dynamic-data exchange (DDE) channel. To free system resources, you should close channels you aren't using.

Argument	Explanation
ChanNum	The channel number of the DDE conversation as returned by **DDEInitiate()**. If *ChanNum* isn't an open channel number, an error occurs.

See also

DDE Statements and Functions

DDEExecute

DDEInitiate()

DDEPoke

DDERequest$()

DDETerminateAll

DDETerminateAll

DDETerminateAll

Closes all dynamic-data exchange (DDE) channels opened by Word; **DDETerminateAll** does not close channels opened to Word by client applications. Using this statement is the same as using a **DDETerminate** statement for each open channel. **DDETerminateAll** does not cause an error if no DDE channels are open.

If you interrupt a macro that opens a DDE channel, you may inadvertently leave a channel open. Open channels are not closed automatically when a macro ends, and each open channel uses system resources. For this reason, it's a good idea to use **DDETerminateAll** while debugging a macro that opens one or more DDE channels.

_____

See also

DDE Statements and Functions

DDEExecute

DDEInitiate()

DDEPoke

DDERequest$()

DDETerminate

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SendKeys Examples

One use of **SendKeys** is to insert text or select items in a Word dialog box (for example, to provide default text or initially select a specific item). This example displays the Open dialog box (File menu) and inserts the default text "TESTFILE" into the File Name box. You can produce the same effect with the single instruction SendKeys "%foTESTFILE".

```
SendKeys "TESTFILE"
Dim dlg As FileOpen
x = Dialog(dlg)
```

The following example adds an annotation to the Word Help Contents screen of Word Help. In the **SendKeys** instruction, c chooses the Contents button, %e selects the Edit menu, a chooses the Annotate command, and a$ inserts the annotation.

```
a$ = "My annotation for Help."
SendKeys "c%ea" + a$
Shell "WINHELP.EXE C:\WINWORD\WINWORD.HLP"
```

The following example starts Microsoft Excel with the worksheet MARCH.XLS and then performs the equivalent to pressing the PAGE DOWN key 20 times:

```
SendKeys "{pgdn 20}"
Shell "C:\EXCEL\MARCH.XLS", 1
```

SendKeys

¬Example

SendKeys *Keys$, Wait*

Sends the keystrokes specified by *Keys$* to the next Word command or active application, just as if they were typed at the keyboard. **SendKeys** must precede instructions that activate the command or application that receives the keystrokes so that those keystrokes are in memory when the macro pauses.

Note

Use **SendKeys** to operate other applications only when there is no alternative, and then use it with caution. In general, dynamic data exchange (DDE) is a better way for Word to interact with other applications because DDE provides a channel for two-way communication between applications and provides a path for detecting and dealing with errors in the other application. You should test even the simplest use of **SendKeys** under a variety of conditions to avoid unpredictable results, data loss, or both.

When stepping through a macro (using the Step or Step Subs button on the Macro toolbar) that contains a **SendKeys** instruction, note that the macro will not perform as expected unless the **SendKeys** instruction and the instruction that activates the target application or dialog box are on the same line, separated by a colon.

Argument	Explanation
Keys$	A key or key sequence, such as "a" for the letter *a*, or "{enter}{pgup}" for the ENTER key followed by the PAGE UP key.
Wait	If Word is not the active application and *Wait* is -1, Word waits for all keys to be processed before proceeding. For example, if you run the following instructions to send *Keys$* to Microsoft Excel, a beep occurs in Word only after text is inserted into all 10 cells. If *Wait* were 0 (zero), the beep would occur as text was inserted into the first cell. AppActivate "Microsoft Excel" For i = 1 To 10 SendKeys "Testing{down}", -1 Next i Beep

To specify keys that do not insert a character, use the following codes.

Key	Code
BACKSPACE	{backspace} or {bs} or {bksp}
BREAK	{break}
CAPS LOCK	{capslock}
CLEAR	{clear}
DEL	{delete} or {del}
DOWN ARROW	{down}
END	{end}
ENTER	{enter} or ~
ESC	{escape} or {esc}

HELP	{help}
HOME	{home}
INS	{insert}
LEFT ARROW	{left}
NUM LOCK	{numlock}
PAGE DOWN	{pgdn}
PAGE UP	{pgup}
PRINT SCREEN	{prtsc}
RIGHT ARROW	{right}
TAB	{tab}
UP ARROW	{up}
F1, F2, F3,...F16	{F1}, {F2}, {F3},...{F16}

Note
In addition to the key codes listed above, you can specify any keycode listed in **ToolsCustomizeKeyboard**. To do so, use the syntax **{***Code***}**. For example, the instruction SendKeys "{13}" sends the key code for ENTER.

To repeat a character, use the syntax **{***Character Number***}**. For example, SendKeys "{s 10}" repeats the letter *s* 10 times. Remember to put a space between the key and the number. To repeat a key whose code already includes braces, add a space and a number within the braces, for example: SendKeys "{enter 10}". (You cannot repeat **ToolsCustomizeKeyboard** key codes in this way.)

To send a key combination that includes SHIFT, ALT, or CTRL, use the following symbols.

To combine with	Precede the key code by	
SHIFT	+	(plus sign)
ALT	%	(percent sign)
CTRL	^	(caret)

For example, "%{enter}" sends the code for ALT+ENTER. You can group keys with parentheses and precede the group with the key code for a SHIFT, ALT, or CTRL key. For example, the code "+(wordbasic)" specifies WORDBASIC (but you can also simply use the uppercase letters WORDBASIC, without using the plus sign). To send a plus sign (+), a percent sign (%), or a caret (^) as literal text, enclose the character in braces. For example, to send a plus sign, use the instruction SendKeys "{+}". You can also use braces to send parentheses.

Note
When sending key combinations that include the ALT key, make it a rule to send lowercase characters. For example, to open the File menu (ALT, F), use "%f." Using "%F" is equivalent to pressing ALT+SHIFT+F.

See also

DDE Statements and Functions

AppActivate

DDEExecute

DDEInitiate()

DDEPoke

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Declare Example

This macro uses the selection as a search keyword in Help. If there is one exact match, the macro displays the topic. If there is more than one match, the macro displays the Search dialog box (Help) with the keyword selected. If there is no match, the macro displays the Search dialog box, with the keyword list scrolled to the keyword closest in spelling to the selected word.

```
Declare Function WinHelp Lib "USER.EXE"(hWnd As Integer, lpHelpFile \
    As String, wCmd As Integer, dwData As String) As Integer
Declare Function GetActiveWindow Lib "USER.EXE"() As Integer
Sub MAIN
    hWnd = GetActiveWindow
    helpFile$ = "C:\WINWORD\WINWORD.HLP"
    wCmd = 261                  'The decimal value for HELP_PARTIALKEY
    keyWord$ = Selection$()
    success = WinHelp(hWnd, helpFile$, wCmd, keyWord$)
    If success = 0 Then MsgBox "Could not start Windows Help"
End Sub
```

Declare

¬Example

Declare Sub *SubName* **Lib** *LibName$* [**(***ArgumentList***)**] [**Alias** *Routine$*]

Declare Function *FunctionName*[**$**] **Lib** *LibName$* [**(***ArgumentList***)**] [**Alias** *Routine$*] **As** *Type*

Makes available a function stored in a Windows dynamic-link library (DLL), a Word add-in library (WLL), or an executable file (EXE) for use as a function or subroutine in a WordBasic macro. The declaration specifies the name of the function, the library file in which it is stored, and any arguments the function takes. **Declare** statements are usually placed before the macro's main subroutine; they cannot be placed inside a subroutine or function.

Windows includes standard function libraries such as USER.EXE and GDI.EXE whose functions are documented in the *Microsoft Windows Programmer's Reference, Volume 2*, available in the Microsoft Windows Software Development Kit or from Microsoft Press. For a list of the appropriate module and library for each function, see *Volume 1*. More information on using these functions is provided in Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

Caution
When experimenting with Windows functions, save your work often. An invalid argument passed to a function could result in unpredictable behavior in Word or other applications.

Argument	Explanation
Sub or **Function**	Use **Function** if the function you are declaring returns a value; use **Sub** if it does not.
SubName or *FunctionName*[**$**]	The name used in the macro to call the function. This name does not have to be the actual name of the function in the library. You can define the actual name using the **Alias** part of the statement. If *FunctionName* returns a string, it should include a dollar sign ($) like other string functions in WordBasic.
LibName$	The filename of the library containing the routine, in quotation marks. Use the entire filename, including the extension, to prevent ambiguity. If the file is not stored in the Windows directory or the SYSTEM subdirectory, include the complete path.
ArgumentList	A list of variables representing arguments that are passed to the function. See the following table for the syntax used within *ArgumentList*.
Alias *Routine$*	The actual name of the function in the library, in quotation marks. It is required only if the name specified after **Sub** or **Function** is not the actual name of the

	function.
As *Type*	Declares the data type of the value returned by a function. The type is one of the following: **As Integer** for an integer or logical (BOOL) return type; **As String** for a string (LPSTR) return type; **As Long** for a long return type; **As Double** for a double return type.

The *ArgumentList* argument has the following syntax:

(*Variable*[*$*] [**As** *Type*] [**,** *Variable*[*$*] [**As** *Type*]] [**,** ...]**)**

The following table describes the parts of *ArgumentList.*

Part	Explanation
Variable[*$*]	A WordBasic variable name. For string variables, adding a dollar sign ($) to the variable name is the same as specifying **As String** --- for example, fileName$ is the same as fileName As String. If there is no **As** *Type*, and the variable name does not end in a dollar sign, the variable defaults to a WordBasic numeric variable (double-precision floating point).
As *Type*	Declares the data type of the argument required by the function: **As Integer** for integer or logical (BOOL) arguments; **As String** (or simply **$** at the end of the variable name) for string (LPSTR) arguments; **As Long** for long arguments; **As Double** for double arguments.

See also

Definition and Declaration Statements and Functions

Dim

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Dim Examples

This example declares three variables to be shared by multiple subroutines and would typically be placed before the Sub MAIN instruction.

```
Dim Shared numDocs, docName$, savePath$
```

The following example stores the names of the days of the week in the array Days$ and displays the third element of the array, "Tuesday," in the status bar:

```
Dim Days$(6)
Days$(0) = "Sunday" : Days$(1) = "Monday" : Days$(2) = "Tuesday"
Days$(3) = "Wednesday" : Days$(4) = "Thursday"
Days$(5) = "Friday" : Days$(6) = "Saturday"
Print Days$(2)
```

The following example declares the variable num as shared, sets the variable to 6, and then calls the MyRoutine subroutine. The subroutine adds 10 to num, and the main subroutine displays the new value. Note that the instruction Dim Shared num must be placed outside both subroutines.

```
Dim Shared num              'Declare num as shared
Sub MAIN
    num = 6                 'Set the value of num
    MyRoutine               'Call the routine
    Print num               'Display the new value of num
End Sub

Sub MyRoutine
    num = num + 10          'Increase the value of num
End Sub
```

In the following example, **Dim** is used to define the dialog record TOVrec in which the values of the View tab in the Options dialog box (Tools menu) are stored. This macro toggles the setting of the Paragraph Marks check box.

```
Dim TOVrec As ToolsOptionsView          'Define dialog record
GetCurValues TOVrec                     'Get the current state
If TOVrec.Paras = 1 Then                'If on
    TOVrec.Paras = 0                    'turn off
Else                                    'Else
    TOVrec.Paras = 1                    'turn on
End If
ToolsOptionsView TOVrec                 'Apply new setting
```

In the following example, **Dim** is used to define the dialog record Dlg for a user-defined dialog box:

```
Begin Dialog UserDialog 320, 57, "A Simple Dialog Box"
    Text 5, 16, 205, 18, "This is a simple dialog box."
    OKButton 220, 4, 88, 21
    CancelButton 220, 29, 88, 21
End Dialog
Dim Dlg As UserDialog               'Define the dialog record
Dialog Dlg                          'Display the dialog box
```

Dim

¬Example

Dim [**Shared**] *Var1* [**(***Size***)**] [**,** *Var2* [**(***Size***)**]] [**,** ...]

Dim *DialogRecord* **As** *DialogName*

Dim *DialogRecord* **As UserDialog**

Declares a variable to be shared by more than one subroutine, defines an array variable, or defines a dialog record. **Dim** statements that declare shared variables must be placed outside the macro's subroutines --- usually before the main subroutine.

Syntax	Purpose
Dim *Var***(***Size***)**	Defines an array variable. *Size* is the subscript of the last element in the array. In WordBasic, the first element in an array is always numbered 0 (zero), so to define an array of 7 elements --- for example, the days of the week --- *Size* would be 6.
Dim Shared *Var*[**(***Size***)**]	Declares a string, numeric, or array variable to be shared by more than one subroutine. The **Dim** statement must be placed outside of any subroutine. Usually, shared variable declarations are placed at the start of a macro, before the main subroutine.
Dim *DialogRecord* **As** *DialogName*	Defines a dialog record variable for a Word dialog box. *DialogName* can be any WordBasic statement that corresponds to a Word command or dialog box. For example, **FormatFont** or **FileOpen** are valid as *DialogName*.
Dim *DialogRecord* **As UserDialog**	Defines a dialog record for a user-defined dialog box.

See also

Definition and Declaration Statements and Functions

Declare

Dialog

Let

Redim

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Let Example

Each of the following instructions assigns the value 100 to the variable A:

```
Let A = 100
A = 100
```

Let

¬Example

[**Let**] *Var* = *Expression*

Assigns the value of an expression to a variable. The use of **Let** is optional.

See also

Definition and Declaration Statements and Functions

Dim

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Redim Examples

This example illustrates how you might use **Redim** to recycle the storage space in an array variable when you have finished using the first set of values. This helps conserve system resources.

```
Dim BigArray$(100)
BigArray$(0) = "A long text string"
'Series of statements that define array elements 1 through 99
BigArray$(100) = "Another long text string"
'Series of statements that make use of the values in BigArray$()
Redim BigArray$(100)
'Series of statements that define and make use of new array elements
```

The following macro retrieves the author name from the Summary Info dialog box (File menu) and the user name from the User Info tab in the Options dialog box (Tools menu), and then compares the two values. If they are different, a message box is displayed.

The **Dim** statement stores the array of values from the Summary Info dialog box in the dialog record dlg. After the author name is retrieved, these values are no longer needed. **Redim** can therefore recycle the dialog record dlg to store the values from the User Info tab (Options dialog box).

```
Sub MAIN
Dim dlg As FileSummaryInfo
GetCurValues dlg
Author$ = dlg.Author
Redim dlg As ToolsOptionsUserInfo
GetCurValues dlg
UserName$ = dlg.Name
If Author$ <> UserName$ Then
    MsgBox "Author does not match user."
End If
End Sub
```

Redim

¬Example

Redim [**Shared**] *Var*[**(***Size***)**] [**,** *Var*[**(***Size***)**]]

Redim *Var* **As** *DialogName*

Redim *Var* **As UserDialog**

Empties an array variable so that the array elements may be redefined. **Redim** stands for "redimension" --- you can specify a new *Size* when redefining an array variable. Note that whenever you use **Redim**, existing array contents are lost. **Redim** can also redefine a dialog record for a Word dialog box or a user-defined dialog box. For descriptions of arguments used with **Redim**, see Dim.

See also

Definition and Declaration Statements and Functions

Dim

Let

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Begin Dialog...End Dialog Example

This example defines and displays a dialog box (shown after the instructions) that includes every dialog box control available. Note that the **Picture** instruction assumes that the file C:\WIN31\FLOCK.BMP exists (FLOCK.BMP is included with Microsoft Windows version 3.1). Once you have defined a dialog box with **Begin Dialog...End Dialog**, you need two additional instructions to display it: a **Dim** instruction that defines a dialog record in which the dialog box's values are stored, and a **Dialog** instruction that displays the dialog box. Note that a dialog function is required to display a document in the **FilePreview** control. For information on dialog functions, see Creating Dynamic Dialog Boxes.

```
Sub MAIN
Dim MyList$(2)
MyList$(0) = "Blue"
MyList$(1) = "Green"
MyList$(2) = "Red"
Begin Dialog UserDialog 612, 226, "Every Dialog Box Control", \
        .DlgFunction
    ComboBox 8, 76, 176, 111, MyList$(), .ComboBox1$
    OKButton 253, 199, 108, 21
    CancelButton 131, 199, 108, 21
    PushButton 10, 199, 108, 21, "Push Button"
    Text 417, 186, 35, 13, "Text"
    TextBox 417, 199, 179, 18, .TextBox1$
    CheckBox 198, 79, 180, 16, "Check Box", .CheckBox1
    DropListBox 417, 5, 179, 108, MyList$(), .DropListBox1
    ListBox 195, 102, 189, 83, MyList$(), .ListBox1
    GroupBox 7, 4, 177, 65, "Group Box"
    OptionGroup  .OptionGroup1
        OptionButton 17, 16, 148, 16, "Option Button 1"
        OptionButton 17, 33, 148, 16, "Option Button 2"
        OptionButton 17, 50, 148, 16, "Option Button 3"
    Picture 199, 7, 181, 62, "C:\WIN31\FLOCK.BMP", 0, .Picture1
    FilePreview 417, 31, 179, 148, .Fileprev
End Dialog
Dim sampleDlg As UserDialog
button = Dialog(sampleDlg)
End Sub


Function DlgFunction(identifier$, action, suppvalue)
    'A dialog function is required to display
    'a document in the FilePreview control.
End Function
```

Every Dialog Box Control
Group Box
First Button
Second Button
Third Button
Check Box
Blue
Green
Red
Blue
Green
Red
Blue
Text
PushButton
Cancel
OK

Begin Dialog...End Dialog

¬Example

Begin Dialog UserDialog [*HorizPos***,** *VertPos***,**] *Width***,** *Height* [**,** *Title$*] [**,** **.***DialogFunction*]
Series of dialog box definition instructions
End Dialog

Encloses the instructions that define a dialog box you create within a macro. A dialog box definition consists of a series of instructions that define different elements of the dialog box, such as the OK button, Cancel button, and so on. Dialog box elements are also known as *dialog box controls*.

The easiest way to create a dialog box is to use the Dialog Editor. With the Dialog Editor, you can use the mouse to design the dialog box. The Dialog Editor then creates the WordBasic code needed to define the dialog box. For information on creating and working with dialog boxes, see Chapter 5, "Working with Custom Dialog Boxes," in the Microsoft Word Developer's Kit.

Note
Every custom dialog box must contain at least one command button so the user can close the dialog box. For that reason, a dialog box definition must include either an **OKButton**, a **CancelButton**, or a **PushButton** instruction; otherwise, a WordBasic error will occur when the macro is run.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the dialog box from the upper-left corner of the Word window, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font. If *HorizPos* and *VertPos* are not specified, Word centers the dialog box within the Word window.
Width, *Height*	The width and height of the dialog box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
Title$	The text that is displayed in the title bar of the dialog box. If *Title$* is not specified, "Microsoft Word" is used.
.*DialogFunction*	The name of a dialog function associated with the dialog box; used for dialog boxes that update dynamically while the dialog box is displayed.

You can use the tab key to move between controls in a dialog box. The order of the instructions in a dialog box definition determines the tabbing order in the dialog box. By default, the dialog control represented by the first instruction in the dialog box definition will be selected when the dialog box is displayed. However, you can override this default by using the **DlgFocus** statement in a dialog function. You can also use the **Dialog** statement to specify the default button for a dialog box.

See also

Dialog Box Definition and Control Statements and Functions

CancelButton

CheckBox
ComboBox
Dialog
Dim
DropListBox
FilePreview
GroupBox
ListBox
OKButton
OptionButton
OptionGroup
Picture
PushButton
Text
TextBox

CancelButton

CancelButton *HorizPos*, *VertPos*, *Width*, *Height* [, .*Identifier*]

Creates a Cancel button in a custom dialog box. A user chooses the Cancel button to close the dialog box without taking any action.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the Cancel button from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the Cancel button, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
.*Identifier*	An optional identifier used by statements in a dialog function that act on the Cancel button, such as **DlgEnable** and **DlgVisible**. If omitted, "Cancel" is the default identifier.

If you use the **Dialog** statement to display the dialog box and the user chooses the Cancel button, WordBasic generates an error, which you can trap with **On Error**.

If you use the **Dialog()** function to display the dialog box and the user chooses the Cancel button, the function returns 0 (zero) rather than generating an error.

To see an example of **CancelButton** in a dialog box definition, see Begin Dialog...End Dialog Example.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

Dialog

Err

Error

OKButton

On Error

PushButton

CheckBox

CheckBox *HorizPos*, *VertPos*, *Width*, *Height*, *Label$*, **.***Identifier*

Creates a check box in a custom dialog box.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the rectangle containing the check box and its associated label from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the check box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
Label$	The label associated with the check box. An ampersand (&) precedes the character in *Label$* that is the access key for selecting and clearing the check box.
.*Identifier*	Together with the dialog record name, **.***Identifier* creates a variable whose value corresponds to the state of the check box. The form for this variable is *DialogRecord***.***Identifier* (for example, dlg.MyCheckBox). *DialogRecord***.***Identifier* can return the following values: 0 (zero) The check box is cleared. 1 The check box is selected. -1 The check box is filled with gray. The identifier string (**.***Identifier* minus the period) is also used by statements in a dialog function that act on the check box, such as **DlgEnable** and **DlgVisible**.

To see an example of **CheckBox** in a dialog box definition, see Begin Dialog...End Dialog Example.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

ComboBox

ComboBox *HorizPos*, *VertPos*, *Width*, *Height*, *ArrayVariable$()*, **.***Identifier*[**$**]

Creates a combo box --- a single control that is a combination of a list box and a text box---in a custom dialog box. A user can either select an item from the list or type a new item in the text box.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the combo box from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the combo box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
ArrayVariable$()	A text array containing the items to be listed in the combo box.
.*Identifier*[**$**]	Together with the dialog record name, **.***Identifier*[**$**] creates a variable whose value corresponds to the text of the item chosen or the text typed in the combo box. The form for this variable is *DialogRecord***.***Identifier*[**$**] (for example, dlg.MyComboBox$). The dollar sign ($) is optional; you can use it to indicate that the variable is a string variable. The identifier string (**.***Identifier*[**$**] minus the period) is also used by statements in a dialog function that act on the combo box, such as **DlgEnable** and **DlgVisible**.

The macro must define and assign values to the elements in *ArrayVariable$()* before defining the dialog box containing a combo box. To see an example of **ComboBox** in a dialog box definition, see Begin Dialog...End Dialog Example.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

Dialog

Dim

DlgListBoxArray

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Dialog Examples

This example uses the **Dialog()** function to display the Font dialog box (Format menu) so the user can set different character formatting options. The instruction `x = Dialog(dlg)` returns a value corresponding to the button the user chooses and removes the need for error handling. The last instruction in the example, `FormatFont dlg`, implements the options that the user selected in the dialog box. Without this statement, nothing happens, even though the dialog box has been displayed and the user has chosen the OK button.

```
Dim dlg As FormatFont
GetCurValues dlg
x = Dialog(dlg)
FormatFont dlg
```

The following example is the same as the previous one, except that it uses the **Dialog** statement to display the dialog box and includes error handling. If the user chooses the Cancel button, the resulting error is trapped and a message is displayed.

```
Dim dlg As FormatFont
GetCurValues dlg
On Error Goto trap
Dialog dlg
FormatFont dlg
Goto skiptrap
trap:
MsgBox "Dialog box closed -- no changes"
skiptrap:
```

The following example displays a simple custom dialog box with text box and command button controls. Since the text box control is the first control defined in the dialog box definition, it has the focus when the dialog box is displayed. The `x = Dialog(dlg, 1)` instruction makes the Find button the default command button (instead of the OK button).

```
Begin Dialog UserDialog 346, 90, "Social Security Macro"
    TextBox 9, 22, 192, 18, .secnum
    Text 9, 6, 183, 13, "Social Security Number:"
    OKButton 249, 4, 88, 21
    CancelButton 249, 28, 88, 21
    PushButton 249, 52, 88, 21, "Find..."
End Dialog
Dim dlg As UserDialog
x = Dialog(dlg, 1)
```

Dialog, Dialog()

¬Example

Dialog *DialogRecord* [**,** *DefaultButton*] [**,** *TimeOut*]

Dialog(*DialogRecord* [**,** *DefaultButton*] [**,** *TimeOut*]**)**

The **Dialog** statement and **Dialog()** function are both used to display the dialog box specified by *DialogRecord* in a preceding **Dim** instruction. The dialog box displayed can be a Word dialog box or one that you define within a macro.

If you use the **Dialog** statement to display a dialog box and the user chooses the Cancel button, Word generates an error that an **On Error** instruction can trap. If you don't want or need to trap the error, you can use the **Dialog()** function to display the dialog box. In addition to displaying the dialog box, the **Dialog()** function returns a value corresponding to the button chosen. The return values are the same as those used to specify *DefaultButton*.

Argument	Explanation
DialogRecord	The name given to the dialog record in the **Dim** instruction.
DefaultButton	The default command button: - 2 No default command button. -1 The OK button. 0 (zero) The Cancel button. > 0 (zero) A command button, where 1 is the first **PushButton** statement in the definition, 2 is the second, and so on. The default command button is chosen if the user presses ENTER when a dialog box control other than a command button has the focus. Unless you specify otherwise, the OK button is the default command button. *DefaultButton* is used only with custom dialog boxes. For *DefaultButton* to have any effect, the instruction for the specified button must be preceded in the dialog box definition by an instruction for a nonbutton dialog box control that can receive the focus, such as a list box or check box. (Note that because a text control cannot receive the focus, it does not meet this criterion.) Otherwise, the first button in the dialog box definition is the default command button. If the dialog box definition has a dialog function associated with it, *DefaultButton* may be overridden by a **DlgFocus$** statement. For information on creating dialog functions, see Creating Dynamic Dialog Boxes.

TimeOut	The amount of time, in milliseconds, the dialog box is displayed. If 0 (zero) or omitted, the dialog box is displayed until the user closes it.

Note

A macro can use any number of custom dialog boxes, but WordBasic supports only one custom dialog record at a time (you can define any number of dialog records for Word dialog boxes). This means that you can't define a series of custom dialog records, intending to use them later in the macro. Each custom dialog record overwrites the previous one. If your macro uses more than one custom dialog box and you need to store the values contained in a dialog record that will be overwritten, save the values in regular variables. For more information about using custom dialog records, see Chapter 5, "Working with Custom Dialog Boxes," in the Microsoft Word Developer's Kit.

_____

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

Dim

GetCurValues

DialogEditor

DialogEditor

Starts the Dialog Editor if it is not already running and makes it the active application. If the Dialog Editor is already running, the **DialogEditor** statement makes it the active application. If the Dialog Editor is shrunk to an icon, the **DialogEditor** statement restores and activates it. For more information about the Dialog Editor, see Chapter 5, "Working with Custom Dialog Boxes," in the Microsoft Word Developer's Kit.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgControlId() Example

This macro displays a simple dialog box containing the OK and Cancel buttons and a check box labeled "Test." The dialog function checks to see if the value for action is 2, meaning a control has been selected or chosen. If so, the dialog function then checks to see if the control selected is the first one defined---in other words, if the numeric identifier returned by **DlgControlId()** is 0 (zero). If it is, a message box is displayed in front of the dialog box, which remains on-screen.

```
Sub MAIN
Begin Dialog UserDialog 320, 50, "Test", .MyDlgFunction
    CheckBox 212, 10, 88, 19, "Test", .Test                    '0
    OKButton 7, 10, 88, 21                                     '1
    CancelButton 109, 10, 88, 21                               '2
End Dialog
Dim dlg As UserDialog
button = Dialog(dlg)
End Sub

Function MyDlgFunction(identifier$, action, suppvalue)
If action = 2 Then
    If DlgControlId(identifier$) = 0 Then
        MsgBox "You selected the first control " + \
                                "defined in the dialog box definition."
    End If
End If
End Function
```

DlgControlId()

¬Example

DlgControlId(*Identifier$***)**

Used within a dialog function to return the numeric identifier for the dialog box control specified by *Identifier$*, the string identifier of the dialog box control. Numeric identifiers are numbers, starting at 0 (zero), that correspond to the positions of dialog box control instructions within a dialog box definition. For example, consider the following instruction in a dialog box definition:

```
CheckBox 97, 54, 36, 12, "&Update", .MyCheckBox
```

The instruction `DlgControlId("MyCheckBox")` returns 0 (zero) if the **CheckBox** instruction is the first instruction in the dialog box definition, 1 if it's the second instruction, and so on.

In most cases, your dialog functions will perform actions based on the string identifier of the control that was selected. However, dialog functions can use numeric identifiers to manipulate more quickly a large number of controls (for example, the number of controls found in a multiple-panel dialog box). Note that if you change the order of instructions in a dialog box definition, the numeric identifiers for the corresponding controls will change also.

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

See also

Dialog Box Definition and Control Statements and Functions

DlgFocus

DlgValue

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgEnable Example

This example disables the dialog box control with the string identifier "Set" when the dialog box is initially displayed. (The main subroutine that contains the dialog box definition is not shown.)

```
Function MyDlgFunction(identifier$, action, suppvalue)
Select Case action
    Case 1                          'The dialog box is displayed
        DlgEnable "Set", 0
    Case 2                          'The user selects a control
        'Statements that perform actions based
        'on which control is selected go here
    Case 3                          'Text change (not applicable)
    Case Else
End Select
End Function
```

DlgEnable, DlgEnable()

¬Example

DlgEnable *Identifier*[*$*] [**,** *On*]

DlgEnable(*Identifier*[*$*]**)**

The **DlgEnable** statement is used within a dialog function to enable or disable the dialog box control identified by *Identifier*[*$*] while the dialog box is displayed. When a dialog box control is disabled, it is visible in the dialog box, but is dimmed and not functional.

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

Argument	Explanation
Identifier[*$*]	The string or numeric identifier of the dialog box control.
On	Enables or disables the dialog box control: 1 Enables the control. 0 (zero) Disables the control. Omitted Toggles the control.

The **DlgEnable()** function returns the following values.

Value	Explanation
-1	If the control is enabled
0 (zero)	If the control is disabled

See also

Dialog Box Definition and Control Statements and Functions

DlgVisible

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgFilePreview Example

This example updates the file preview box according to the selection made in a list box containing filenames. (The main subroutine that contains the dialog box definition is not shown.)

```
Function PreviewFiles(identifier$, action, suppvalue)
If action = 1 Then                              'Preview first file in list
    previewfilename$ = DlgText$("ListBox1")
    DlgFilePreview$ previewfilename$
ElseIf action = 2 Then                          'The user selects a control
    If identifier$ = "ListBox1" Then                    'Preview selected file
        previewfilename$ = DlgText$("ListBox1")
        DlgFilePreview$ previewfilename$
    End If
End If
End Function
```

DlgFilePreview, DlgFilePreview$()

¬Example

DlgFilePreview [*Identifier*[*$*]] [**,** *Filename$*]

DlgFilePreview$()

The **DlgFilePreview** statement is used within a dialog function to preview the document specified by *Filename$* (which does not have to be open). If you don't specify *Filename$*, **DlgFilePreview** previews the active document. The document is displayed in the box created with a **FilePreview** instruction in the dialog box definition. Because a custom dialog box can have only one file preview box, the use of *Identifier*[*$*] to specify the file preview box is optional.

The **DlgFilePreview$()** function returns the path and filename of the document displayed in the file preview box. If the active document has not been saved as a file (for example, a new document such as Document2 or a macro open for editing), **DlgFilePreview$()** returns the window name instead of the path and filename.

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

See also

Dialog Box Definition and Control Statements and Functions

DlgSetPicture

FilePreview

Picture

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgFocus Example

This example sets the focus on the control "MyControl1" when the dialog box is initially displayed. (The main subroutine that contains the dialog box definition is not shown.)

```
Function MyDlgFunction(identifier$, action, suppvalue)
Select Case action
    Case 1                          'The dialog box is displayed
        DlgFocus "MyControl1"
    Case 2                          'The user selects a control
        'Statements that perform actions based
        'on which control is selected go here
    Case 3                          'Text change (not applicable)
    Case Else
End Select
End Function
```

DlgFocus, DlgFocus$()

¬Example

DlgFocus *Identifier*[*$*]

DlgFocus$()

The **DlgFocus** statement is used within a dialog function to set the focus on the dialog box control identified by *Identifier*[*$*] while the dialog box is displayed. When a dialog box control has the focus, it is active and responds to keyboard input. For example, if a text box has the focus, any text you type appears in that text box.

The **DlgFocus$()** function returns the string identifier for the dialog box control that currently has the focus.

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

See also

Dialog Box Definition and Control Statements and Functions

DlgEnable

DlgVisible

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgListBoxArray Example

This example changes the contents of a list box while the dialog box is displayed. The dialog box definition in the main subroutine (not shown) defines a dialog box containing a list box, the OK and Cancel buttons, and a Change button. When the user chooses the Change button, the **If** control structure in Case 2 of the dialog function takes over: A new string array is defined, **DlgListBoxArray** is used to replace the contents of the list box, and the Change button is disabled. By default, a custom dialog box is closed when the user chooses a command button such as the Change button; however, in this example, the keepDisplayed = 1 and dlgTest = keepDisplayed instructions keep the dialog box displayed (when a dialog function returns a value of 1, the dialog box remains displayed).

```
Function dlgTest(identifier$, action, suppvalue)
Select Case action
    Case 1                          'The dialog box is displayed
    Case 2                          'The user selects a control
        If identifier$ = "Change" Then
                Dim MyArray2$(1)
                MyArray2$(0) = "New first item"
                MyArray2$(1) = "New second item"
                DlgListBoxArray "MyListBox", MyArray2$()
                DlgFocus "MyListBox"
                DlgEnable "Change", 0
                keepDisplayed = 1
        End If
    Case 3                          'Text change (not applicable)
    Case Else
End Select
dlgTest = keepDisplayed
End Function
```

DlgListBoxArray, DlgListBoxArray()

¬Example

DlgListBoxArray *Identifier*[*$*]**,** *ArrayVariable$()*

DlgListBoxArray(*Identifier*[*$*] [**,** *ArrayVariable$()*]**)**

The **DlgListBoxArray** statement is used within a dialog function to fill a list box, drop-down list box, or combo box with the contents of *ArrayVariable$()*. You can use this statement to change the contents of a list box, drop-down list box, or combo box in a custom dialog box while the dialog box is displayed.

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

Argument	Explanation
Identifier[*$*]	The string or numeric identifier of the dialog box control. The control must be a list box, drop-down list box, or combo box.
ArrayVariable$()	A string array containing items to be displayed in the specified list box, drop-down list box, or combo box.

The **DlgListBoxArray()** function fills *ArrayVariable$()* with the contents of the list box, drop-down list box, or combo box specified by *Identifier*[*$*] and returns the number of entries in the list box, drop-down list box, or combo box. *ArrayVariable$()* is optional with the **DlgListBoxArray()** function; if *ArrayVariable$()* is omitted, **DlgListBoxArray()** returns the number of entries in the specified control.

See also

Dialog Box Definition and Control Statements and Functions

DlgEnable

DlgFocus

DlgText

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgSetPicture Example

In this example, the user selects a person's name from a list box and a picture of the person is displayed in the dialog box. The instructions assume that the graphics file to display is in the C:\PHOTOS directory and that the filename is the same as the selected name, followed by a .PIC filename extension. (The main subroutine that contains the dialog box definition is not shown.)

```
Function dlgTest(identifier$, action, suppvalue)
Select Case action
    Case 1                          'The dialog box is displayed
    Case 2                          'The user selects a control
        If identifier$ = "Name" Then
                name$ = DlgText$("Name")
                pictureName$ = "C:\PHOTOS\" + name$ + ".PIC"
                DlgSetPicture "Photo", pictureName$, 0
        End If
    Case 3                          'Text change (not applicable)
    Case Else
End Select
End Function
```

DlgSetPicture

¬Example

DlgSetPicture *Identifier*[*$*]**,** *PictureName$***,** *PictureType*

Used in a dialog function to set the graphic displayed by a **Picture** dialog box control whose identifier is *Identifier*[*$*].

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

Argument	Explanation
Identifier[*$*]	The string or numeric identifier of the dialog box control.
PictureName$	The name of the graphics file, AutoText entry, or bookmark containing the graphic to be displayed. If the picture does not exist, Word displays the text "(missing picture)" within the rectangle that defines the picture area.
PictureType	A value indicating how the graphic is stored: 0 (zero) *PictureName$* is a file. 1 *PictureName$* is an AutoText entry. 2 *PictureName$* is a bookmark. 3 *PictureName$* is ignored; the graphic is retrieved from the Clipboard.

See also

Dialog Box Definition and Control Statements and Functions

Picture

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgText Example

This example hides and displays a control in a dialog box, and, accordingly, toggles the label of a command button between "Hide Control" and "Show Control." (The main subroutine that contains the dialog box definition is not shown.) By default, a custom dialog box is closed when the user chooses a command button; however, in this example, the `keepDisplayed = 1` and `dlgTest = keepDisplayed` instructions keep the dialog box displayed (when a dialog function returns a value of 1, the dialog box remains displayed).

```
Function dlgTest(identifier$, action, suppvalue)
Select Case action
    Case 1                          'The dialog box is displayed
    Case 2                          'The user chooses a button
        If identifier$ = "Hide" Then
                If DlgText$("Hide") = "Hide &Control" Then
                        DlgVisible "Option1", 0
                        DlgText$ "Hide", "Show &Control"
                Else
                        DlgVisible "Option1", 1
                        DlgText$ "Hide", "Hide &Control"
                End If
                keepDisplayed = 1
        End If
    Case 3                          'Text change (not applicable)
    Case Else
End Select
dlgTest = keepDisplayed
End Function
```

DlgText, DlgText$()

¬Example

DlgText *Identifier*[*$*]**,** *Text$*

DlgText$(***Identifier*[*$*])**

The **DlgText** statement is used in a dialog function to set the text or label for the dialog box control identified by *Identifier*[*$*]. The **DlgText** statement does not change the string identifier of a dialog box control.

For a text box or combo box, **DlgText** sets the text within the text box. For dialog box controls that have labels, such as check boxes, option buttons, option groups, and command buttons, **DlgText** sets the label. For list boxes, **DlgText** sets the selection to *Text$* or to the first item that begins with *Text$*.

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

Argument	Explanation
Identifier[*$*]	The string or numeric identifier of the dialog box control
Text$	The text or label to set

The **DlgText$()** function returns the text or label of the dialog box control identified by *Identifier*[*$*]. If the dialog box control is a list box, the text of the selected item is returned. If the dialog box control is a text box or combo box, **DlgText$()** returns the text that appears in the text box.

See also

Dialog Box Definition and Control Statements and Functions

DlgValue

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgUpdateFilePreview Example

This example updates a file preview box after the user has selected a left indent measurement for the selected text. After the user selects the measurement, the file preview box shows the change in the indent. Note that the instruction Val(DlgText$("DropListBox1") is necessary to remove the string "inch" from the measurement; **FormatParagraph** does not accept "inch" as a valid measurement.

```
Sub MAIN
Dim indents$(3)
indents$(0) = ".5 inch" : indents$(1) = "1 inch"
indents$(2) = "1.5 inches" : indents$(3) = "2 inches"
Begin Dialog UserDialog 354, 258, "Change Indents", .dlgTest
    PushButton 255, 10, 88, 21, "Close", .close
    Text 15, 8, 151, 25, "Select a left indent:", .Text1
    FilePreview 15, 58, 219, 183, .fileprev
    DropListBox 15, 29, 179, 30, indents$(), .DropListBox1
End Dialog
Dim dlg As UserDialog
x = Dialog(dlg)
End Sub

Function dlgTest(identifier$, action, suppvalue)
Select Case action
Case 1                                  'The dialog box is displayed
Case 2                                  'The user selects a control
    If identifier$ = "DropListBox1" Then
        indent$ = Str$(Val(DlgText$("DropListBox1")))
        FormatParagraph .LeftIndent = indent$
        DlgUpdateFilePreview
    End If
Case Else
End Select
End Function
```

DlgUpdateFilePreview

¬Example

DlgUpdateFilePreview [*Identifier$*]

Used within a dialog function to update a file preview box (created by a **FilePreview** instruction in a dialog box definition). Since a dialog box can have only one file preview box, *Identifier$* is optional.

_____

See also

Dialog Box Definition and Control Statements and Functions

DlgFilePreview

FilePreview

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DlgValue Example

This example is the portion of a dialog function that runs when the check box with the identifier "MyCheckBox" is selected. **DlgValue()** is used to determine whether the check box was turned on or off. If it was turned on, the instructions redefine an array and fill a list box with the items in the array.

```
Case 2                                  'The user chooses a button
    If identifier$ = "MyCheckBox" Then
        If DlgValue("MyCheckBox") = 1 Then
                ReDim MyArray$(1)
                MyArray$(0) = "Alberto"
                MyArray$(1) = "Gina"
                DlgListBoxArray "MyListBox", MyArray$()
        End If
    End If
```

DlgValue, DlgValue()

¬Example

DlgValue *Identifier*[*$*]**,** *Value*

DlgValue(*Identifier*[*$*]**)**

The **DlgValue** statement is used in a dialog function to select or clear a dialog box control by setting the numeric value associated with the control specified by *Identifier*[*$*]. For example, DlgValue "MyCheckBox", 1 selects a check box, DlgValue "MyCheckBox", 0 clears a check box, and DlgValue "MyCheckBox", -1 fills the check box with gray. An error occurs if *Identifier*[*$*] specifies a dialog box control such as a text box or an option button that cannot be set with a numeric value.

For information about using a dialog function, see Creating Dynamic Dialog Boxes.

Argument	Explanation
Identifier[*$*]	The string or numeric identifier of the dialog box control. The **DlgValue** statement and **DlgValue()** function can be used with identifiers created by the following statements: **CheckBox**, **ComboBox**, **DropListBox**, **ListBox**, and **OptionGroup**.
Value	Numeric value to set the dialog box control identified by *Identifier$*

The **DlgValue()** function returns the numeric value of the check box, list box, drop-down list box, combo box, or option group identified by *Identifier*[*$*]. For example, DlgValue("MyCheckBox") returns 1 if the check box is selected, 0 (zero) if it is cleared, and -1 if it is filled with gray. The instruction DlgValue("MyListBox") returns 0 (zero) if the first item in the list box is selected, 1 if the second item is selected, and so on.

See also

Dialog Box Definition and Control Statements and Functions

DlgText

DlgVisible, DlgVisible()

DlgVisible *Identifier*[*$*] [**,** *On*]

DlgVisible(*Identifier*[*$*]**)**

The **DlgVisible** statement is used in a dialog function to make visible or to hide the dialog box control identified by *Identifier*[*$*]. You can use **DlgVisible** to hide little-used options until they are needed, or to create dialog boxes with multiple panels of controls.

Argument	Explanation
Identifier[*$*]	The string or numeric identifier of the dialog box control.
On	Shows or hides the dialog box control: 1 Shows the control. 0 (zero) Hides the control. Omitted Toggles the control between visible and hidden.

The **DlgVisible()** function returns the following values.

Value	Explanation
-1	If the control is visible
0 (zero)	If the control is hidden

For an example that uses **DlgVisible**, see DlgText Example.

See also

Dialog Box Definition and Control Statements and Functions

DlgEnable

DlgFocus

DlgText

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DropListBox Example

See ListBox Example for an example of a list box. To modify the example so that it displays a drop-down list box instead of a list box, change the **ListBox** instruction to **DropListBox**, and change the *Height* argument to the height of the drop-down list box when it is dropped down.

For an example of **DropListBox** used in a complete dialog box definition, see Begin Dialog...End Dialog Example.

DropListBox

¬Example

DropListBox *HorizPos*, *VertPos*, *Width*, *Height*, *ArrayVariable$()*, **.***Identifier*

Creates a drop-down list box from which a user can select an item within a user-defined dialog box. **DropListBox** has the same syntax as **ListBox**. When a drop-down list box is dropped down, the portion that is dropped down can cover other controls in the dialog box or fall outside the dialog box.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the drop-down list box from the upper-left corner of the dialog box in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the list box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font. *Height* is the height of the list box when it is dropped down.
ArrayVariable$()	A string array containing the items to be listed, one list box item per array element.
.*Identifier*	Together with the dialog record name, creates a variable whose value corresponds to the selected list box item. The form for this variable is *DialogRecord***.***Identifier* (for example, dlg.MyList). The identifier string (**.***Identifier* minus the period) is also used by statements in a dialog function that act on the list box, such as **DlgEnable** and **DlgVisible**.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

Combo Box

DlgListBoxArray

ListBox

FilePreview

FilePreview *HorizPos*, *VertPos*, *Width*, *Height*, **.***Identifier*

Creates a file preview box --- similar to the one in the Find File dialog box (File menu) --- in a custom dialog box. A custom dialog box can have only one file preview box.

You cannot use the Dialog Editor to create a **FilePreview** instruction. A good work around is to use another control as a stand-in. In the Dialog Editor, size and position the stand-in control as you want the file preview box to be sized and positioned. Then, after you paste the dialog box definition into your macro, convert the stand-in instruction to a **FilePreview** instruction.

The file preview box displays the document that is active when the dialog box is displayed. You can also control which document is displayed in the file preview box by using **DlgFilePreview** in a dialog function. For more information about using a dialog function, see Creating Dynamic Dialog Boxes.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the file preview box from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the file preview box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
.*Identifier*	An identifier used by a **DlgFilePreview** instruction in a dialog function to change the document displayed in the file preview box.

For an example, see DlgUpdateFilePreview Example.

See also

Dialog Box Definition and Control Statements and Functions

DlgFilePreview

DlgUpdateFilePreview

Picture

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetCurValues Example

This example uses **GetCurValues** to retrieve the date the active document was created from the Document Statistics dialog box (Summary Info command, File menu). The instructions then use date functions to calculate the number of days since the document was created and display a message box according to the result.

```
Dim dlg As DocumentStatistics
GetCurValues dlg
docdate$ = dlg.Created
age = Now() - DateValue(docdate$)
age = Int(age)
Select Case age
    Case 0
        MsgBox "This document is less than a day old."
    Case Is > 0
        MsgBox "This document was created" + Str$(age) + " day(s) ago."
    Case Else
        MsgBox "Check your computer's date and time."
End Select
```

For an example that uses **GetCurValues** and shows how to toggle any check box, see Abs() Example.

GetCurValues

¬Example

GetCurValues *DialogRecord*

Stores in *DialogRecord* the current values for a previously defined dialog record for a Word dialog box. You use **Dim** to define a dialog record (for example, Dim dlg As ToolsOptionsSave), **GetCurValues** to store the settings (for example, GetCurValues dlg), and the syntax *DialogRecord***.***ArgumentName* to return specific settings (for example, save = dlg.FastSaves). You do not need to use **GetCurValues** with a dialog record for a custom dialog box; if you do, the instruction has no effect.

See also

Dialog Box Definition and Control Statements and Functions

Dialog

Dim

GroupBox

GroupBox *HorizPos*, *VertPos*, *Width*, *Height*, *Label$* [, .*Identifier*]

Creates a box in a custom dialog box that you can use to enclose a group of related option buttons or check boxes.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance from the upper-left corner of the group box to the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the group box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
Label$	The label displayed in the upper-left corner of the group box. An ampersand (&) precedes the character in *Label$* that is the access key for moving to the group box.
.*Identifier*	An optional identifier used by statements in a dialog function that act on the group box. For example, you can use this identifier with **DlgText** to change *Label$* while the dialog box is displayed.

For an example, see OptionGroup Example.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

DlgText

OptionGroup

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InputBox$() Examples

This example asks the user to type a word, which is assigned to the variable word$. You could use this variable in a macro that counts the occurrences of a given word in the active document (as in the example for **EditFindFound()**).

```
word$ = InputBox$("Count instances of:", "WordCounter")
```

The following example prompts the user to type a number from 1 to 10. **InputBox$()** returns what the user types as a string, which the **Val()** function converts to a number to assign to the numeric variable num. Note that if the user spells out "Ten" instead of typing "10," **Val()** returns 0.

```
num = Val(InputBox$("Type a number from 1 to 10:"))
```

This example asks the user to type two lines, which are assigned to the variable a$. The **InStr()** function checks for a newline character; if one exists, the **Left$()** and **Right$()** functions are used to redefine a$ without the newline character.

```
a$ = InputBox$("Type two lines:")
newline = InStr(a$, Chr$(11))
If newline <> 0 Then
    a$ = Left$(a$, newline - 1) + Right$(a$, Len(a$) - newline)
End If
```

InputBox$()

¬Example

InputBox$(***Prompt$* [,** *Title$*] [**,** *Default$*]**)**

Displays a dialog box requesting a single piece of information and returns the text entered in the dialog box when the user chooses the OK button. If the user chooses the Cancel button, an error occurs. You can use the **On Error** statement to trap the error.

Argument	Explanation
Prompt$	Text displayed in the dialog box indicating the kind of information requested.
Title$	Text displayed in the title bar of the dialog box (if omitted, Word uses the title "Microsoft Word").
Default$	Text that initially appears in the text box of the dialog box. This value is returned if the user types nothing before choosing OK. If *Default$* is longer than 255 characters, an error occurs.

If the user presses ENTER to start a new line while the dialog box is displayed, **InputBox$()** returns text containing Chr$(11) + Chr$(13) (a newline character followed by a paragraph mark) where the line break was made. For a method of removing the newline character, see the third example for this entry.

See also

Dialog Box Definition and Control Statements and Functions

Input

MsgBox

On Error

Val()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ListBox Example

This macro displays a list of menu names. First, it creates an array of menu names to be presented in a list box, then it defines a dialog box containing the list box, and finally it displays the dialog box.

```
Sub MAIN
Dim MyList$(CountMenus(1) - 1)
For i = 1 To CountMenus(1)
    MyList$(i - 1) = MenuText$(1, i)
Next i
On Error Resume Next
Begin Dialog UserDialog 320, 118, "List Box Example"
    Text 27, 8, 129, 13, "This is a list box:"
    ListBox 29, 25, 160, 84, MyList$(), .MyList
    OKButton 226, 5, 88, 21
    CancelButton 226, 29, 88, 21
End Dialog
Dim dlg As UserDialog
Dialog dlg
End Sub
```

ListBox

¬Example

ListBox *HorizPos*, *VertPos*, *Width*, *Height*, *ArrayVariable$()*, **.***Identifier*

Creates a list box from which a user can select an item in a custom dialog box.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance from the upper-left corner of the list box to the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the list box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
ArrayVariable$()	A string array containing the list, one list box item per array element.
.*Identifier*	Together with the dialog record name, **.***Identifier* creates a variable whose value corresponds to the number of the selected list box item. The form for this variable is *DialogRecord***.***Identifier* (for example, dlg.MyList). The identifier string (**.***Identifier* minus the period) is also used by statements in a dialog function that act on the list box, such as **DlgEnable** and **DlgVisible**.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

Combo Box

DlgListBoxArray

DropListBox

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MsgBox Examples

In this example, the **MsgBox** instruction displays a message box containing the message "Unable to find file," the title "MyTest Macro," an OK button, and a Stop symbol (0 + 16 + 0 = 16):

```
MsgBox "Unable to find file.", "MyTest Macro", 16
```

In the following macro, the **If** conditional checks whether or not there is a selection before proceeding. If there is no selected text, the **MsgBox()** function displays a message box asking if the user wants to continue anyway. The second **If** conditional tests the return value. If it's 0 (zero), which means the second button (No) was chosen, the macro ends without running subsequent instructions.

```
Sub MAIN
'Series of instructions that select text
If SelType() <> 2 Then
    button = MsgBox("There is no selection. Continue anyway?", 36)
    If button = 0 Then Goto bye
End If
'Series of instructions that act on the selection
bye:
End Sub
```

MsgBox, MsgBox()

¬Example

MsgBox *Message$* [**,** *Title$*] [**,** *Type*]

MsgBox(*Message$* [**,** *Title$*] [**,** *Type*]**)**

The **MsgBox** statement displays a message in a message box. You can also display a message with the **MsgBox()** function, which returns a value according to the command button the user chooses in the message box. Use **MsgBox()** if you need your macro to take action based on the user's response.

Argument	Explanation
Message$	The message to be displayed in the message box. If *Message$* is longer than 255 characters, an error occurs.
Title$	The title of the message box. If omitted, "Microsoft Word" is the default title.
Type	A value representing the symbol and buttons displayed in the box.

Type is the sum of three values, one from each of the following groups.

Group	Value	Meaning
Button	0 (zero)	OK button (default)
	1	OK and Cancel buttons
	2	Abort, Retry, and Ignore buttons
	3	Yes, No, and Cancel buttons
	4	Yes and No buttons
	5	Retry and Cancel buttons
Symbol	0 (zero)	No symbol (default)
	16	Stop symbol
	32	Question symbol
	48	Attention symbol
	64	Information symbol
Button action	0 (zero)	First button is the default
	256	Second button is the default
	512	Third button is the default

By specifying *Type* as -1, -2, or -8, you can display the message in the status bar instead of a message box. This is similar to using the **Print** statement, but gives you more control over how long the message is displayed. Specifying *Type* as -1 displays the message until another message replaces it; specifying -2 displays the message until a mouse or keyboard action occurs; and specifying -8 displays the message in the entire status bar width until a mouse or keyboard action occurs.

Because the **MsgBox** statement does not return a value, the use of button values other than 0 (zero) is not recommended. To make use of buttons other than the OK button, use the **MsgBox()** function. **MsgBox()** returns the following values.

Return value	Button chosen	Button text
-1	First (leftmost) button	OK

		Yes
		Abort
0 (zero)	Second button	Cancel
		No
		Retry
1	Third button	Cancel
		Ignore

If *Type* is a negative value---that is, if you use **MsgBox()** to display a message in the status bar---**MsgBox()** always returns 0 (zero).

_____

See also

Dialog Box Definition and Control Statements and Functions

InputBox$()

Print

OKButton

OKButton *HorizPos*, *VertPos*, *Width*, *Height* [, .*Identifier*]

Creates an OK button in a custom dialog box. If the user chooses the OK button, the dialog box closes and its settings are applied.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the OK button from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the OK button, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
.*Identifier*	An optional identifier used by statements in a dialog function that act on the OK button, such as **DlgEnable** and **DlgVisible**. If omitted, "OK" is the default identifier.

Note that choosing the OK button in a custom dialog box always returns -1 and choosing the Cancel button always returns 0 (zero). For example, in a dialog box displayed with the instruction `x = Dialog(dlg)`, choosing OK sets the value of `x` to -1. The return value of other buttons is determined by the relative order of the **PushButton** statements in the dialog box definition.

For an example of **OKButton** used in a complete dialog box definition, see Begin Dialog...End Dialog Example.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

CancelButton

PushButton

OptionButton

OptionButton *HorizPos*, *VertPos*, *Width*, *Height*, *Label$* [, **.***Identifier*]

Creates an option button in a custom dialog box. An **OptionGroup** instruction is required for each series of related **OptionButton** instructions, which must be positioned directly following the **OptionGroup** instruction. Within a group of option buttons, only one button can be selected at a time.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the rectangle containing both the option button and its associated text from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the rectangle, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
Label$	The label associated with the option button. An ampersand (&) precedes the character in *Label$* that is the access key for selecting the option button.
.*Identifier*	An optional identifier used by statements in a dialog function that act on the option button. If omitted, the default identifier is the first two words in *Label$* (or the entire string if *Label$* is only one word). You should specify **.***Identifier* because the identifier you assign remains the same regardless of changes you make to *Label$*.

For an example of **OptionButton** used in a complete dialog box definition, see OptionGroup Example.

See also

Dialog Box Definition and Control Statements and Functions

CheckBox

GroupBox

OptionGroup

PushButton

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OptionGroup Example

This macro creates the dialog box shown following it and then inserts either a page break or column break according to the option button the user selects. The identifier defined for the option group is .Brk. Note how the second **If** instruction performs an action based on the value of the dlg.Brk variable.

```
Sub MAIN
Begin Dialog UserDialog 292, 78, "Example"
    OKButton 188, 14, 88, 21
    CancelButton 188, 38, 88, 21
    GroupBox 12, 6, 164, 60, "Break"
    OptionGroup .Brk
        OptionButton 22, 23, 117, 16, "&Page Break"
        OptionButton 22, 41, 133, 16, "&Column Break"
End Dialog
Dim dlg As UserDialog
If Dialog(dlg) Then
    If dlg.Brk = 0 Then InsertPageBreak Else InsertBreak .Type = 1
End If
End Sub
```

OptionGroup

¬Example

OptionGroup ***.Identifier***

Defines a series of related option buttons in a dialog box definition. An **OptionGroup** instruction is required for each series of related **OptionButton** instructions, which must be positioned directly following the **OptionGroup** instruction. Within the group, only one option button can be selected at a time. The **OptionGroup** identifier returns a value corresponding to the selected option button.

Argument	Explanation
.Identifier	Together with the dialog record name, ***.Identifier*** creates a variable whose value corresponds to the selected option button, where 0 (zero) is the first option button in the group, 1 is the second, and so on. The form for this variable is *DialogRecord**.Identifier*** (for example, dlg.Brk). The identifier string (***.Identifier*** minus the period) is also used by statements in a dialog function that act on the option group, such as **DlgEnable** and **DlgVisible**.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

GroupBox

ListBox

OptionButton

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Picture Example

This example defines a square area at the upper-left corner of the dialog box and inserts the graphic stored in the AutoText entry "Smiley." The dialog box control is identified as .Graphic. For an example of a complete dialog box definition, see Begin Dialog...End Dialog Example.

```
Picture 7, 7, 50, 50, "Smiley", 1, .Graphic
```

Picture

¬Example

Picture *HorizPos*, *VertPos*, *Width*, *Height*, *PictureName$*, *Type*, *.Identifier*

Displays a graphic in a custom dialog box. Word automatically sizes the graphic to fit within the *Width* and *Height* you have specified.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the rectangle containing the graphic from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the rectangle, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
PictureName$	The name of the graphics file, AutoText entry, or bookmark containing the graphic that is initially displayed. You can reset *PictureName$* with **DlgSetPicture** in a dialog function when the dialog box is displayed.
Type	A value indicating how the graphic is stored: 0 (zero) *PictureName$* is a graphics file. Paths are allowed. 1 *PictureName$* is an AutoText entry. The AutoText entry must contain a single graphic only (no text or paragraph marks) and must be stored either in NORMAL.DOT or the template attached to the active document. 2 *PictureName$* is a bookmark in the active document. The bookmark must mark a single graphic only (no text or paragraph marks). 3 *PictureName$* is ignored and the graphic is retrieved from the Clipboard. Include empty quotation marks ("") in place of *PictureName$*. The Clipboard must contain a single graphic only. If the Clipboard contains any text from a Word document, the Microsoft Word icon is displayed in the dialog box.
.Identifier	An identifier used by statements in a dialog function that act on the graphic. For example, you can use this identifier with **DlgSetPicture** to display a different graphic.

Note

If the specified picture does not exist, Word displays the text "(missing picture)" within the rectangle that defines the picture area. You can change this behavior by adding 16 to the value for *Type*. If *Type* is 16, 17, 18, or 19 and the specified picture does not exist, Word displays an error (which may be trapped with **On Error**) and displays neither the text "(missing picture)" nor a rectangle in the dialog box.

See also

Dialog Box Definition and Control Statements and Functions

DlgSetPicture

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PushButton Example

This example creates a command button with the text "Create Link," whose identifier is "Link." For an example of **PushButton** used in a complete dialog box definition, see Begin Dialog...End Dialog Example.

```
PushButton 40, 20, 80, 18, "&Create Link", .Link
```

PushButton

¬Example

PushButton *HorizPos*, *VertPos*, *Width*, *Height*, *Label$* [, **.***Identifier*]

Creates a command button in a custom dialog box. When the command button is chosen, the **Dialog()** instruction that displays the dialog box (for example, button = Dialog(dlg)) returns a value corresponding to the relative order of that **PushButton** instruction among the other **PushButton** instructions: 1 for the first command button, 2 for the second, and so on.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the command button from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the command button, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
Label$	The label associated with the command button. An ampersand (&) precedes the character in *Label$* that is the access key for choosing the command button.
.*Identifier*	An optional identifier used by statements in a dialog function that act on the command button. If omitted, the default identifier is the first two words in *Label$* (or the entire string if *Label$* is only one word). You should specify **.***Identifier* because the identifier you assign remains the same regardless of changes you make to *Label$*.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

CancelButton

OKButton

OptionButton

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Text Examples

In this example, the **Text** instruction results in the text "Sample Text":

```
Begin Dialog UserDialog 320, 84, "Sample Dialog Macro"
    Text 10, 6, 160, 12, "&Sample Text", .myTextControl
    TextBox 10, 21, 236, 18, .TextBox1
    OKButton 9, 58, 88, 21
    CancelButton 115, 58, 88, 21
End Dialog
```

The following example displays the first 255 characters of the current selection in the text label.

```
textlabel$ = Left$(Selection$(), 255)
Begin Dialog UserDialog 621, 251, "Show Selection Text"
    OKButton 409, 220, 88, 21
    CancelButton 509, 220, 88, 21
    Text 10, 6, 586, 203, textlabel$, .Text1
End Dialog
```

Text

¬Example

Text *HorizPos*, *VertPos*, *Width*, *Height*, *Label$* [, .*Identifier*]

Creates a rectangle containing fixed text in a custom dialog box. If you want to use the text control to define an access key for a list box or text box, the **Text** instruction must precede the instruction for the list box or text box control.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the text control rectangle from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the rectangle, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
Label$	The label to display in the dialog box. An ampersand (&) precedes the character in *Label$* that is the access key for the control following the **Text** instruction.
.*Identifier*	An optional identifier used by statements in a dialog function that act on the text control. For example, you can use this identifier with **DlgText** to change *Label$* while the dialog box is displayed.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

TextBox

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TextBox Example

In this example, text entered in the multiple-line text box created with the **TextBox** instruction is stored in the dlg.TextBox1 variable:

```
Begin Dialog UserDialog 290, 152, "Sample Dialog Macro"
    Text 10, 6, 260, 12, "&Sample Text: Press ENTER"
    Text 10, 22, 143, 12, "to start a new line."
    TextBox 10, 43, 236, 64, .TextBox1, 1
    OKButton 10, 117, 88, 21
    CancelButton 117, 117, 88, 21
End Dialog
Dim dlg As UserDialog
Dialog dlg
```

TextBox

¬Example

TextBox *HorizPos*, *VertPos*, *Width*, *Height*, **.***Identifier*[**$**] [, *MultiLine*]

Creates a single-line or multiple-line text box into which the user can enter information in a custom dialog box. You can insert as many as 255 characters into a text box control.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance of the upper-left corner of the text box from the upper-left corner of the dialog box, in increments of 1/8 (for *HorizPos*) and 1/12 (for *VertPos*) of the system font.
Width, *Height*	The width and height of the text box, in increments of 1/8 (for *Width*) and 1/12 (for *Height*) of the system font.
.*Identifier*[**$**]	Together with the dialog record name, **.***Identifier*[**$**] creates a variable whose value corresponds to the text in the text box. The form for this variable is *DialogRecord***.***Identifier*[**$**] (for example, dlg.MyTextBox$). The dollar sign ($) is optional; you can use it to indicate that the variable is a string variable. The identifier string (**.***Identifier*[**$**] minus the period) is also used by statements in a dialog function that act on the text box, such as **DlgEnable** and **DlgVisible**.
MultiLine	Specifies whether the text box is a single-line or multiple-line text box: 0 (zero) Single-line text box 1 Multiple-line text box In a multiple-line text box, the user can press ENTER or SHIFT+ENTER to begin new lines when typing text into the box. Note that multiple-line text boxes do not contain scroll bars; however, the user may scroll through text using the arrow keys. If the user presses ENTER or SHIFT+ENTER to begin a new line, the string returned by the identifier will contain a paragraph mark.

See also

Dialog Box Definition and Control Statements and Functions

Begin Dialog...End Dialog

Text

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ChDefaultDir Example

This example creates the directory C:\BAK and then sets AUTOSAVE-PATH to that directory for the current session. The instruction On Error Resume Next prevents an error from stopping the macro if the directory already exists.

```
On Error Resume Next
MkDir "C:\BAK"
Err = 0
ChDefaultDir "C:\BAK", 5
```

ChDefaultDir

¬Example

ChDefaultDir *Path$*, *Type*

Sets one of the Word default directories to the specified path. Unlike **ToolsOptionsFileLocations**, which saves default directory changes in WINWORD6.INI, **ChDefaultDir** changes default directories for the current Word session only. Changes made with **ChDefaultDir** are not reflected on the File Locations tab in the Options dialog box (Tools menu).

Argument	Explanation
Path$	The path to which you want to set the default directory specified by *Type*
Type	A number corresponding to the default directory to set: 0 DOC-PATH 1 PICTURE-PATH 2 USER-DOT-PATH 3 WORKGROUP-DOT-PATH 4 INI-PATH 5 AUTOSAVE-PATH 6 TOOLS-PATH 7 CBT-PATH 8 STARTUP-PATH 15 The style-gallery-template path Note that types 9 through 14 cannot be set with **ChDefaultDir**. To return their values for the current Word session, use **DefaultDir$()**.

For more information about the default directories described here, see WINWORD6.INI Options in Word Readme Help.

See also

Disk Access and Management Statements and Functions

ChDir

DefaultDir$()

Files$()

GetDirectory$()

ToolsOptionsFileLocations

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ChDir Examples

This example changes the directory and then displays the Open dialog box so that you can open a document stored in that directory. You could create a macro like this for a directory you use often and assign the macro to a toolbar button for quick access.

```
ChDir "C:\WINWORD\LETTERS\PERSONAL"
Dim dlg As FileOpen
button = Dialog(dlg)
If button <> 0 Then FileOpen dlg
```

The following example determines whether the current directory is C:\WINWORD and changes to it if it is not:

```
If Files$(".") <> "C:\WINWORD" Then ChDir "C:\WINWORD"
```

ChDir

¬Example

ChDir *Path$*

Sets the current directory to the drive and directory specified by *Path$*. If the drive is omitted, the search for the specified path starts at the current directory. You can use **ChDir** to set the directory so that you do not have to specify the complete path when you use **FileOpen** to open a document.

See also

Disk Access and Management Statements and Functions

Connect

CountDirectories()

Files$()

GetDirectory$()

MkDir

RmDir

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Connect Example

This example establishes a connection to a network drive protected with the password "smiley" and assigns the network drive to the next available network drive letter:

```
Connect .Path = "\\PROJECT\INFO", .Password = "smiley"
```

Connect

¬Example

Connect [**.Drive =** *number,*] **.Path =** *text* [**, .Password =** *text*]

Establishes a connection to a network drive.

Argument	Explanation
.Drive	A number corresponding to the letter you want to assign to the network drive, where 0 (zero) corresponds to the first available drive letter, 1 to the second available drive letter, and so on. If you don't specify .Drive, the next available letter is used.
.Path	The path for the network drive (for example, "\\PROJECT\INFO").
.Password	The password, if the network drive is protected with a password.

See also

Disk Access and Management Statements and Functions

ChDir

CountDirectories()

GetDirectory$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CopyFile Examples

This example copies the document JULY.DOC to the C:\WINWORD\MEMOS directory:

```
CopyFile .FileName = "JULY.DOC", .Directory = "C:\WINWORD\MEMOS"
```

This example displays a dialog box that prompts for the directory to which to copy the active document:

```
a$ = FileName$()
Dim dlg As CopyFile
dlg.FileName = a$
dlg.Directory = "C:\BAK"
x = Dialog(dlg)
If x = - 1 Then CopyFile dlg
```

CopyFile

¬Example

CopyFile .FileName = *text***, .Directory =** *text*

Copies a file to the specified directory. Like the MS-DOS COPY command, **CopyFile** allows you to specify a new name when copying the file.

Argument	Explanation
.FileName	The name of the file to copy. If you do not specify the path, Word copies the file from the current directory.
.Directory	The path of the directory to which the file is copied. Include a filename if you want to rename the file in addition to copying it.

See also

Disk Access and Management Statements and Functions

FileSaveAs

Kill

Name

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountDirectories() Example

This example determines whether there are any subdirectories in the C:\WINWORD directory:

```
dirNum = CountDirectories("C:\WINWORD")
If dirNum = 0 Then
    MsgBox "No subdirectories."
Else
    MsgBox "There are" + Str$(dirNum) + " subdirectories."
End If
```

CountDirectories()

¬Example

CountDirectories(*Directory$***)**

Returns the number of subdirectories contained within *Directory$*. If *Directory$* is omitted, the current directory is assumed. If *Directory$* does not exist, **CountDirectories()** returns -1.

_____

See also

Disk Access and Management Statements and Functions

Files$()

GetDirectory$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DefaultDir$() Example

This example sets TOOLS-PATH to C:\TOOLS if it is not set already:

```
If DefaultDir$(6) = "" Then
    ChDefaultDir "C:\TOOLS", 6
End If
```

DefaultDir$()

¬Example

DefaultDir$(*Type*)

Returns the path currently set for the default directory specified by *Type*.

Argument	Explanation
Type	A number corresponding to a default directory: 0 DOC-PATH 1 PICTURE-PATH 2 USER-DOT-PATH 3 WORKGROUP-DOT-PATH 4 INI-PATH 5 AUTOSAVE-PATH 6 TOOLS-PATH 7 CBT-PATH 8 STARTUP-PATH 9 The program-file path (PROGRAMDIR) 10 The graphics-filter path (fixed; cannot be changed) 11 The text-converter path (fixed; cannot be changed) 12 The proofing-tool path (fixed; cannot be changed) 13 The temporary-file path (set with the MS-DOS TEMP environment variable) 14 The current directory 15 The style-gallery-template path (can be changed using **ChDefaultDir**)

For more information about the default directories described here, see WINWORD6.INI Options in Word Readme Help.

See also

Disk Access and Management Statements and Functions

ChDefaultDir

ChDir

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Files$() Examples

This example returns the first file ending with the .DOC filename extension found in the current directory:

```
a$ = Files$("*.DOC")
```

The following example returns the current path (for example, "C:\WINWORD"):

```
CurDir$ = Files$(".")
```

If NOTE.DOC exists in the current directory, the following example opens the document. If NOTE.DOC does not exist, a message box is displayed.

```
If Files$("NOTE.DOC") <> "" Then
    FileOpen "NOTE.DOC"
Else
    MsgBox "File not found in current directory."
End If
```

The following example fills an array with the names of all files in the current directory. The instructions first count the files to determine the size of the array. Then they define the array, fill it with the filenames, and sort the elements. You could use this array to present a list of files in a user-defined dialog box or to open and perform an operation on each file in the array.

```
temp$ = Files$("*.*")
count = - 1
While temp$ <> ""
    count = count + 1
    temp$ = Files$()
Wend
If count > -1 Then
    Dim list$(count)
    list$(0) = Files$("*.*")
    For i = 1 To count
        list$(i) = Files$()
    Next i
    SortArray list$()
    MsgBox Str$(count + 1) + " files; 1st in array is " + list$(0)
Else
    MsgBox "No files in current directory."
End If
```

Files$()

¬Example

Files$(***FileSpec$*)**

Returns the first filename that matches the file specification *FileSpec$*. If you specify a period (.) as the *FileSpec$*, **Files$()** returns the current path. You can use the MS-DOS wildcard characters --- the asterisk (*) and the question mark (?) --- to specify files.

Argument	Explanation
FileSpec$	The file specification. If you omit the file specification, the next file that matches the most recently used *FileSpec$* is returned.

By specifying *FileSpec$* on the first iteration and omitting it thereafter, you can use **Files$()** to generate a list of files that match *FileSpec$* (as in the final example in this reference entry). If no files match, an empty string ("") is returned.

See also

Disk Access and Management Statements and Functions

FileFind

GetAttr()

GetAttr(*Filename$***)**

Returns a number corresponding to the file attributes set for *Filename$*, the path and filename of the file on which you want information. The attributes correspond to those you can set using the Properties command (File menu) in File Manager. For definitions of the attributes, display the Properties dialog box and choose the Help button. The values returned by **GetAttr()** are additive. For example, if both Read Only and Archive are selected, the return value is 33 (the sum of 1 and 32).

Value	Explanation
0 (zero)	None of the file attributes are selected.
1	The Read Only attribute is selected.
2	The Hidden attribute is selected.
4	The System attribute is selected.
32	The Archive attribute is selected.

For an example, see SetAttr Example.

See also

Disk Access and Management Statements and Functions

SetAttr

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetDirectory$() Example

This example defines an array with the names of the subdirectories in the current directory. In a dialog box definition, you could specify this array in a **ListBox** instruction to present a list of the subdirectories.

```
Dim subdirs$(CountDirectories())
subdirs$(0) = "[..]"
For i = 1 To CountDirectories()
    subdirs$(i) = LCase$(GetDirectory$(i))
Next
```

GetDirectory$()

¬Example

GetDirectory$(**[*Directory$*,**] *Count***)**

Returns the name of the subdirectory specified by *Count* within the specified directory.

Argument	Explanation
Directory$	The path of the directory containing the subdirectory; if omitted, the current directory is assumed.
Count	A number in the range 1 to **CountDirectories()**, where 1 corresponds to the first subdirectory, 2 to the second, and so on.

See also

Disk Access and Management Statements and Functions

CountDirectories()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Kill Example

This example deletes a file in C:\WORD\LETTERS:

```
Kill "C:\WORD\LETTERS\DRAFT.DOC"
```

Kill

¬Example

Kill *Filename$*

Deletes the specified file. If *Filename$* is an open document, an error occurs.

Argument	Explanation
Filename$	The path and filename of the file to delete. If you do not specify the path, Word attempts to delete the file from the current directory.

See also

Disk Access and Management Statements and Functions

CopyFile

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MkDir Examples

This example creates the directory "OUTPUT" by specifying a path. Note that the TEST directory must already exist; **MkDir** cannot create both directories at the same time.

```
MkDir "C:\TEST\OUTPUT"
```

The following example creates a subdirectory within the current directory:

```
MkDir "OUTPUT"
```

The following example creates a directory at the same level as the current directory:

```
MkDir "..\OUTPUT"
```

MkDir

¬Example

MkDir *Name$*

Creates the directory specified by *Name$*. An error occurs if the directory already exists. If you don't specify a path, *Name$* is interpreted relative to the current directory.

_____

See also

Disk Access and Management Statements and Functions

ChDir

CountDirectories()

Files$()

GetDirectory$()

Name

RmDir

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

RmDir Examples

This example deletes all files from a directory and then deletes the directory:

```
Kill "C:\WORD\FCCPROJ\*.*"
RmDir "C:\WORD\FCCPROJ"
```

The following example deletes a subdirectory within the current directory:

```
RmDir "FCCPROJ"
```

The following example deletes a directory at the same level as the current directory:

```
RmDir "..\FCCPROJ"
```

RmDir

¬Example

RmDir *Name$*

Removes the specified directory. *Name$* can be a full or relative path. For this statement to work, the directory must contain no files or subdirectories. **RmDir** cannot remove the current directory.

_____

See also

Disk Access and Management Statements and Functions

ChDir

Files$()

Kill

MkDir

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetAttr Example

This example makes the file C:\TEST\TEST.DOC read-only without affecting the file's other attributes. First, **GetAttr()** returns the current *Attribute* value. Then the **If** conditional uses the MOD operator to determine whether the *Attribute* value is odd, which means the file's Read Only attribute is already selected. In that case, Word displays a message box. If *Attribute* has an even value, the **SetAttr** instruction sets the Read Only attribute.

```
attribs = GetAttr("C:\TEST\TEST.DOC")
Print attribs
If attribs MOD 2 Then
    MsgBox "File is already read-only."
Else
    SetAttr "C:\TEST\TEST.DOC", 1
End If
```

SetAttr

¬Example

SetAttr *Filename$*, *Attribute*

Sets file attributes for *Filename$*. The attributes correspond to those you can set using the Properties command (File menu) in File Manager. For definitions of the attributes, display the Properties dialog box and choose the Help button. The file whose attributes you want to set must be closed; an error occurs if you try to set an attribute for an open document.

Argument	Explanation
Filename$	The path and filename of the file whose attribute you want to set
Attribute	Specifies the attribute: 0 (zero) Clears all attributes except System 1 Read Only 2 Hidden 32 Archive

Note

Attribute values are additive. For example, to set both the Read Only and Archive attributes, *Attribute* would be 33 (the sum of 1 and 32). For a way to set the Read Only attribute independently, see the example for this entry. You cannot set the System attribute in combination with the Archive attribute.

See also

Disk Access and Management Statements and Functions

GetAttr()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawAlign Example

This example left aligns all drawing objects within the current paragraph. Using the predefined bookmark "\Para," **DrawSetRange** sets the drawing object range to the paragraph containing the insertion point. For more information about predefined bookmarks, see Predefined Bookmarks.

```
DrawSetRange "\Para"
For count = 1 To DrawCount()
    DrawExtendSelect count
Next count
DrawAlign .Horizontal = 1, .Vertical = 0, .RelativeTo = 0
```

DrawAlign

¬Example

DrawAlign [**.Horizontal =** *number*] [**, .Vertical =** *number*] [**, .RelativeTo =** *number*]

Aligns the selected drawing objects. The arguments for the **Align** statement correspond to the options in the Align dialog box (Align Drawing Objects button, Drawing toolbar).

Argument	Explanation
.Horizontal	Specifies a horizontal alignment: 0 (zero) None; existing horizontal positions are preserved 1 Left 2 Center 3 Right
.Vertical	Specifies a vertical alignment: 0 (zero) None; existing vertical positions are preserved 1 Top 2 Center 3 Bottom
.RelativeTo	Specifies what the objects are aligned with: 0 (zero) Each other 1 Page

See also

Drawing Statements and Functions

DrawCount()

DrawExtendSelect

DrawSelect

DrawSetRange

FormatDrawingObject

DrawArc

DrawArc

Switches to page layout view and adds a default arc drawing object in front of the current text layer. A default arc is shaped and oriented like the lower-left portion of a circle and is inserted at the upper-left corner of the current page.

_____

See also

Drawing Statements and Functions

DrawEllipse

DrawFlipHorizontal

DrawFlipVertical

DrawGetType()

DrawLine

DrawRotateLeft

DrawRotateRight

FormatDrawingObject

DrawBringForward

DrawBringForward

Moves the selected drawing object in front of the next one in a stack of drawing objects. **DrawBringForward** does not move a drawing object from the layer behind text to the layer in front of text. An error occurs if the selection is not a drawing object.

_____

See also

Drawing Statements and Functions

DrawBringInFrontOfText

DrawBringToFront

DrawSendBackward

DrawSendBehindText

DrawSendToBack

DrawBringInFrontOfText

DrawBringInFrontOfText

Moves the selected drawing object from the layer behind text to the layer in front of text. The object is placed behind any other drawing objects already in front of the text layer. An error occurs if the object is already part of the drawing layer above text or if the selection is not a drawing object.

_____

See also

Drawing Statements and Functions

DrawBringForward

DrawBringToFront

DrawSendBackward

DrawSendBehindText

DrawSendToBack

DrawBringToFront

DrawBringToFront

Moves the selected drawing object to the topmost position in a stack of drawing objects. **BringToFront** does not move a drawing object from the layer behind text to the layer in front of text. An error occurs if the selection is not a drawing object.

_____

See also

Drawing Statements and Functions

DrawBringForward

DrawBringInFrontOfText

DrawSendBackward

DrawSendBehindText

DrawSendToBack

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawCallout Example

This example inserts and formats a callout that points to the left. The **DrawCallout** instruction inserts a default callout whose type is set by the **FormatCallout** instruction. The **FormatDrawingObject** instruction then positions and sizes the callout and adds a fill color. Coordinates for the text box within the callout are defined in the two-dimensional array calloutpts$() and then applied using **DrawSetCalloutTextbox**. Finally, text is inserted into the callout.

```
DrawCallout
FormatCallout .Type = 1, .Gap = "10 pt", .Angle = 4, \
    .Drop = "Center", .Length = "100 pt", .Border = 1, \
    .AutoAttach = 0, .Accent = 0
FormatDrawingObject .FillColor = 0, .FillPatternColor = 7, \
    .FillPattern = 5, .ArrowStyle = 2, \
    .HorizontalPos = "173 pt", .HorizontalFrom = 1, \
    .VerticalPos = "100 pt", .VerticalFrom = 1, \
    .Height = "42 pt", .Width = "161 pt"
Dim calloutpts$(2, 2)
calloutpts$(1, 1) = "63 pt"
calloutpts$(1, 2) = "0 pt"
calloutpts$(2, 1) = "97 pt"
calloutpts$(2, 2) = "42 pt"
DrawSetCalloutTextbox calloutpts$()
Insert "This callout points to the left."
```

DrawCallout

¬Example

DrawCallout

Inserts a callout drawing object in front of the current text layer at the upper-left corner of the document. A callout consists of a text box and a line segment within a bounding rectangle. When you first insert a callout, the text box is the same size as the bounding rectangle, so the line segment does not appear. You change the size of the text box with the **DrawSetCalloutTextbox** statement.

To change the size and position of the rectangle bounding the callout, as well as the fill color and line style, use **FormatDrawingObject**. You can change other options for the callout with **FormatCallout**.

See also

Drawing Statements and Functions

DrawGetCalloutTextbox

DrawSetCalloutTextbox

DrawTextBox

FormatCallout

FormatDrawingObject

DrawClearRange

DrawClearRange

Clears a drawing range. For more information about drawing ranges, see DrawSetRange.

See also

Drawing Statements and Functions

DrawSetRange

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawCount() Example

This example displays a message box showing the number of drawing objects in the active document:

```
DrawSetRange "\Doc"
numobjects = DrawCount()
MsgBox "There are" + Str$(numobjects) + " drawing objects" + \
        " in the document."
```

DrawCount()

¬Example

DrawCount()

Returns the number of drawing objects whose anchors are in the range set by **DrawSetRange**.

_____

See also

Drawing Statements and Functions

DrawGetType()

DrawSetRange

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawCountPolyPoints() Example

This example determines whether the first two drawing objects in the document are freeform shapes. If they are, the points of the first freeform drawing object are loaded into the array pointsArray() and applied to the second freeform drawing object.

```
DrawSetRange "\Doc"
If DrawGetType(1) = 7 And DrawGetType(2) = 7 Then
    size = DrawCountPolyPoints(1)
    Dim pointsArray(size, 2)
    DrawGetPolyPoints pointsArray(), 1
    DrawSetPolyPoints size, pointsArray(), 2
Else
    MsgBox "The first or second drawing object is " + \
        "not a freeform shape."
End If
```

DrawCountPolyPoints()

¬Example

DrawCountPolyPoints([*Object*]**)**

Returns the number of points in a freeform drawing object. If the specified object is not a freeform shape, an error occurs.

Argument	Explanation
Object	Specifies a drawing object: Omitted — The selected drawing object > 0 (zero) — An object whose anchor is in a range set by the **DrawSetRange** statement, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs.

See also

Drawing Statements and Functions

DrawFreeformPolygon

DrawGetPolyPoints

DrawSetPolyPoints

DrawDisassemblePicture

DrawDisassemblePicture

Converts the selected graphic into a group of drawing objects. If the selected graphic cannot be disassembled, Word inserts it into a text box drawing object. If a graphic is not selected, an error occurs.

See also

Drawing Statements and Functions

DrawGroup

DrawUngroup

DrawEllipse

DrawEllipse

Switches to page layout view and adds a default elliptical drawing object in front of the current text layer. A default elliptical drawing object is a circle and is inserted at the upper-left corner of the current page.

_____

See also

Drawing Statements and Functions

DrawGetType()

DrawRoundRectangle

FormatDrawingObject

DrawExtendSelect

DrawExtendSelect *Count*

Selects the drawing object specified by *Count*, whose anchor is within the drawing range set by the **DrawSetRange** statement. *Count* is the position of the drawing object relative to the text layer: 1 corresponds to the object closest to the text layer, 2 to the next object, and so on. If one or more drawing objects are already selected, the drawing object specified by *Count* is added to the group of selected objects. If no drawing range has been set, or if *Count* is greater than the number of objects whose anchors are within the range, an error occurs.

_____

See also

Drawing Statements and Functions

DrawGroup

DrawSelect

DrawFlipHorizontal

DrawFlipHorizontal

Flips the selected drawing object from left to right. **DrawFlipHorizontal** works only on objects drawn using the Drawing toolbar; an error occurs if the selected object is an embedded object.

_____

See also

Drawing Statements and Functions

DrawFlipVertical

DrawRotateLeft

DrawRotateRight

DrawFlipVertical

DrawFlipVertical

Flips the selected drawing object from top to bottom. **DrawFlipVertical** works only on objects drawn using the Drawing toolbar; an error occurs if the selected object is an embedded object.

See also

Drawing Statements and Functions

DrawFlipHorizontal

DrawRotateLeft

DrawRotateRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawFreeformPolygon Example

This example adds an N-shaped drawing object in front of the text layer by inserting a default freeform drawing object and then using **DrawSetPolyPoints** to change it. Because the lines in an N shape have four end points, eight elements must be assigned to the pointsArray$() array (four pairs of two values). End points are measured relative to the upper-left corner of the current page.

```
DrawFreeformPolygon
Dim pointsArray$(4, 2)
pointsArray$(1, 1) = "150 pt"
pointsArray$(1, 2) = "200 pt"
pointsArray$(2, 1) = "150 pt"
pointsArray$(2, 2) = "100 pt"
pointsArray$(3, 1) = "200 pt"
pointsArray$(3, 2) = "200 pt"
pointsArray$(4, 1) = "200 pt"
pointsArray$(4, 2) = "100 pt"
DrawSetPolyPoints 4, pointsArray$()
```

DrawFreeformPolygon

¬Example

DrawFreeformPolygon

Switches to page layout view and adds a default freeform drawing object in front of the text layer. A default freeform shape consists of a single line segment and is inserted at the upper-left corner of the current page. To change the freeform shape, you use the **DrawSetPolyPoints** statement.

See also

Drawing Statements and Functions

DrawCountPolyPoints()

DrawGetPolyPoints

DrawSetPolyPoints

FormatDrawingObject

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawGetCalloutTextbox Example

This example displays a message box showing the horizontal position, vertical position, width, and height of the text area within the selected callout:

```
Dim pts$(2, 2)
DrawGetCalloutTextbox pts$()
msg$ = "HorizPos: " + pts$(1, 1) + Chr$(13)
msg$ = msg$ + "VertPos :" + pts$(1, 2) + Chr$(13)
msg$ = msg$ + "Width :" + pts$(2, 1) + Chr$(13)
msg$ = msg$ + "Height :" + pts$(2, 2) + Chr$(13)
MsgBox msg$, "Text Area In Callout"
```

DrawGetCalloutTextbox

¬Example

DrawGetCalloutTextbox *TwoDimensionalArray*[**$**]*()* [**,** *Object*]

Fills a two-dimensional array with coordinates that describe the position and size of the text area within the bounding rectangle of the specified callout drawing object.

Argument	Explanation
TwoDimensionalArray[**$**]*()*	The predefined two-dimensional array to fill with the coordinates of the specified callout text box. *TwoDimensionalArray*[**$**]*()* may be a string or numeric array. A string array is filled with text measurements consisting of a number followed by the abbreviation for the default measurement unit. A numeric array is filled with the numbers only. For example, given an array pts() defined with the instruction Dim pts(2, 2), the instruction DrawGetCalloutTextbox pts() fills the array as follows: pts(1, 1) The horizontal position (the distance from the left edge of the bounding rectangle to the text area), in the default measurement unit pts(1, 2) The vertical position (the distance from the top edge of the bounding rectangle to the text area) pts(2, 1) The width of the text area pts(2, 2) The height of the text area
Object	Specifies a drawing object: Omitted The selected drawing object > 0 (zero) An object whose anchor is in a range set by the **DrawSetRange** statement, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs. If the specified drawing object is not a callout, an error occurs.

See also

Drawing Statements and Functions

DrawCallout

DrawSetCalloutTextbox

DrawSetInsertToTextbox

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawGetPolyPoints Example

This example stores the coordinates of the selected freeform drawing object in the array pointsArray(). If a freeform drawing object is not selected, an error occurs.

```
If DrawGetType() = 7 Then
    size = DrawCountPolyPoints()
    Dim pointsArray(size, 2)
    DrawGetPolyPoints pointsArray()
Else
    MsgBox "The selected drawing object is not a freeform shape."
End If
```

For another example, see DrawCountPolyPoints() Example.

DrawGetPolyPoints

¬Example

DrawGetPolyPoints *TwoDimensionalArray*[**$**]*()* [**,** *Object*]

Fills a two-dimensional array with coordinates of the end points in the specified freeform drawing object. If the specified object is not a freeform shape, an error occurs. You can use the array as an argument for **DrawSetPolyPoints** to apply the coordinates to another freeform drawing object. You must define the array before running **DrawGetPolyPoints**; use **DrawCountPolyPoints()** to determine the appropriate size for the array.

Argument	Explanation
TwoDimensionalArray[**$**]*()*	The predefined two-dimensional array to fill with the coordinates of the specified freeform drawing object. *TwoDimensionalArray*[**$**]*()* may be a string or numeric array. A string array is filled with text measurements consisting of a number followed by the abbreviation for the default measurement unit. A numeric array is filled with the numbers only.
Object	Specifies a drawing object: Omitted — The selected drawing object > 0 (zero) — An object whose anchor is in a range set by the **DrawSetRange** statement, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs.

See also

Drawing Statements and Functions

DrawCountPolyPoints()

DrawFreeformPolygon

DrawSetPolyPoints

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawGetType() Example

This example sets the drawing range to the entire document using the predefined bookmark "\Doc," counts the number of arcs in the document, and then displays the result in a message box:

```
DrawSetRange "\Doc"
numarcs = 0
For i = 1 To DrawCount()
    type = DrawGetType(i)
    If type = 6 Then numarcs = numarcs + 1
Next
MsgBox "The document contains" + Str$(numarcs) + " arc(s)."
```

DrawGetType()

¬Example

DrawGetType([*Count*]**)**

Returns a number corresponding to the type of the drawing object specified by *Count*. *Count* is in the range 1 to **DrawCount()**, the number of objects within a range specified by **DrawSetRange**. If *Count* is omitted, the type of the selected drawing object is returned.

DrawGetType() returns the following values.

Value	Explanation
0 (zero)	*Count* is not specified and a drawing object is not selected.
1	*Count* is not specified and more than one drawing object is selected.
2	The drawing object is a line.
3	The drawing object is a text box.
4	The drawing object is a rectangle.
5	The drawing object is an ellipse.
6	The drawing object is an arc.
7	The drawing object is a freeform shape.
8	The drawing object is a callout.

See also

Drawing Statements and Functions

DrawCount()

DrawSelect

DrawSetRange

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawGroup Example

This example sets the drawing range to the current paragraph and then selects each object whose anchor is in the range. If the number of objects in the range is greater than one, they are grouped together.

```
DrawSetRange "\Para"
numobjects = DrawCount()
For i = 1 To numobjects
    DrawExtendSelect i
Next i
If numobjects > 1 Then DrawGroup
```

DrawGroup

¬Example

DrawGroup

Groups the selected drawing objects so they can be manipulated as a single object.

_____

See also

Drawing Statements and Functions

DrawDisassemblePicture

DrawExtendSelect

DrawSelect

DrawSetRange

DrawUngroup

DrawInsertWordPicture

DrawInsertWordPicture

Opens a temporary document (a Word Picture object) and displays the Picture and Drawing toolbars. When the user closes the document or clicks Close Picture on the Picture toolbar, the object is embedded in the document that was active when **DrawInsertWordPicture** was run. The instruction DrawInsertWordPicture is the same as the instruction InsertObject .Class = "Word.Picture.6".

See also

Drawing Statements and Functions

InsertDrawing

InsertObject

InsertPicture

DrawLine

DrawLine

Adds a default linear drawing object in front of the current text layer. The default line segment is inserted at the upper-left corner of the current page.

_____

See also

Drawing Statements and Functions

DrawArc

DrawFlipHorizontal

DrawFlipVertical

DrawFreeformPolygon

DrawGetType()

DrawRotateLeft

DrawRotateRight

FormatDrawingObject

DrawNudgeDown

DrawNudgeDown

Moves the selected drawing object or objects down 7.5 points (72 points = 1 inch) or, if Snap To Grid is selected in the Snap To Grid dialog box (Drawing toolbar), moves the selected objects down by the measurement in the Vertical Spacing box.

_____

See also

Drawing Statements and Functions

DrawNudgeDownPixel

DrawNudgeLeft

DrawNudgeRight

DrawNudgeUp

DrawSnapToGrid

DrawNudgeDownPixel

DrawNudgeDownPixel

Moves the selected drawing object or objects down one pixel (the smallest unit on the screen).

_____

See also

Drawing Statements and Functions

DrawNudgeDown

DrawNudgeLeftPixel

DrawNudgeRightPixel

DrawNudgeUpPixel

DrawNudgeLeft

DrawNudgeLeft

Moves the selected drawing object or objects left 7.5 points (72 points = 1 inch) or, if Snap To Grid is selected in the Snap To Grid dialog box (Drawing toolbar), moves the selected objects left by the measurement in the Horizontal Spacing box.

_____

See also

Drawing Statements and Functions

DrawNudgeDown

DrawNudgeLeftPixel

DrawNudgeRight

DrawNudgeUp

DrawSnapToGrid

DrawNudgeLeftPixel

DrawNudgeLeftPixel

Moves the selected drawing object or objects left one pixel (the smallest unit on the screen).

_____

See also

Drawing Statements and Functions

DrawNudgeDownPixel

DrawNudgeLeft

DrawNudgeRightPixel

DrawNudgeUpPixel

DrawNudgeRight

DrawNudgeRight

Moves the selected drawing object or objects right 7.5 points (72 points = 1 inch) or, if Snap To Grid is selected in the Snap To Grid dialog box (Drawing toolbar), moves the selected objects right by the measurement in the Horizontal Spacing box.

_____

See also

Drawing Statements and Functions

DrawNudgeDown

DrawNudgeLeft

DrawNudgeRightPixel

DrawNudgeUp

DrawSnapToGrid

DrawNudgeRightPixel

DrawNudgeRightPixel

Moves the selected drawing object or objects right one pixel (the smallest unit on the screen).

_____

See also

Drawing Statements and Functions

DrawNudgeDownPixel

DrawNudgeLeftPixel

DrawNudgeRight

DrawNudgeUpPixel

DrawNudgeUp

DrawNudgeUp

Moves the selected drawing object or objects up 7.5 points (72 points = 1 inch) or, if Snap To Grid is selected in the Snap To Grid dialog box (Drawing toolbar), moves the selected objects up by the measurement in the Vertical Spacing box.

_____

See also

Drawing Statements and Functions

DrawNudgeDown

DrawNudgeLeft

DrawNudgeRight

DrawNudgeUpPixel

DrawSnapToGrid

DrawNudgeUpPixel

DrawNudgeUpPixel

Moves the selected drawing object or objects up one pixel (the smallest unit on the screen).

_____

See also

Drawing Statements and Functions

DrawNudgeDownPixel

DrawNudgeLeftPixel

DrawNudgeRightPixel

DrawNudgeUp

DrawRectangle

DrawRectangle

Adds a default rectangular drawing object in front of the current text layer. A default rectangle is a square and is inserted at the upper-left corner of the current page.

_____

See also

Drawing Statements and Functions

DrawFreeformPolygon

DrawRoundRectangle

DrawTextBox

FormatDrawingObject

DrawReshape

DrawReshape

Toggles the handles on the selected freeform drawing object between the bounding rectangle (where you use the handles to scale or resize the object) and the vertices of the freeform shape (where you use the handles to reshape the object). You can select multiple freeform drawing objects before running **DrawReshape**. If a freeform drawing object is not selected, an error occurs.

See also

Drawing Statements and Functions

DrawFreeformPolygon

DrawGetType()

DrawSelect

DrawRotateLeft

DrawRotateLeft

Rotates the selected drawing object counterclockwise 90 degrees. **DrawRotateLeft** works only on objects drawn using the Drawing toolbar; an error occurs if the selected object is an embedded object.

See also

Drawing Statements and Functions

DrawFlipHorizontal

DrawFlipVertical

DrawRotateRight

DrawRotateRight

DrawRotateRight

Rotates the selected drawing object clockwise 90 degrees. **DrawRotateRight** works only on objects drawn using the Drawing toolbar; an error occurs if the selected object is an embedded object.

See also

Drawing Statements and Functions

DrawFlipHorizontal

DrawFlipVertical

DrawRotateLeft

DrawRoundRectangle

DrawRoundRectangle

Adds a default rectangular drawing object with rounded corners in front of the current text layer. A default rounded rectangle has sides of equal length and is inserted at the upper-left corner of the current page.

_____

See also

Drawing Statements and Functions

DrawEllipse

DrawFreeformPolygon

DrawRectangle

DrawTextBox

FormatDrawingObject

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawSelect Example

This example selects the last drawing object in the document and brings it to the topmost position among the drawing objects on the same drawing layer:

```
DrawSetRange "\Doc"
numobjects = DrawCount()
DrawSelect numobjects
DrawBringToFront
```

DrawSelect, DrawSelect()

¬Example

DrawSelect *Object*

DrawSelect(*Object***)**

The **DrawSelect** statement selects the specified drawing object and deselects any other selected drawing objects. To select a drawing object without deselecting other objects, use the **DrawExtendSelect** statement. The **DrawSelect()** function behaves the same as the statement and also returns -1 if the specified object was selected.

Argument	Explanation
Object	Specifies a drawing object whose anchor is in a range set by **DrawSetRange**, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs.

See also

Drawing Statements and Functions

DrawExtendSelect

DrawSetRange

DrawSelectNext

DrawSelectNext

Selects the next drawing object, where "next" means next closest to the top in a stack of drawing objects. If the topmost object is selected, Word selects the bottommost object. If a drawing object is not selected when **DrawSelectNext** runs, an error occurs. Note that you do not need to set a drawing range with **DrawSetRange** to use **DrawSelectNext**.

See also

Drawing Statements and Functions

DrawExtendSelect

DrawSelect

DrawSelectPrevious

DrawSelectPrevious

DrawSelectPrevious

Selects the previous drawing object, where "previous" means next closest to the bottom in a stack of drawing objects. If the bottommost object is selected, Word selects the topmost object. If a drawing object is not selected when **DrawSelectPrevious** runs, an error occurs. Note that you do not need to set a drawing range with **DrawSetRange** to use **DrawSelectPrevious**.

See also

Drawing Statements and Functions

DrawExtendSelect

DrawSelect

DrawSelectNext

DrawSendBackward

DrawSendBackward

Moves the selected drawing object behind the previous one in a stack of drawing objects. **DrawSendBackward** does not move a drawing object from the layer in front of text to the layer behind text. An error occurs if a drawing object is not selected.

_____

See also

Drawing Statements and Functions

DrawBringForward

DrawBringInFrontOfText

DrawBringToFront

DrawSendBehindText

DrawSendToBack

DrawSendBehindText

DrawSendBehindText

Moves the selected drawing object from the layer in front of text to the layer behind text. The object is placed in front of any other drawing objects already behind the text layer. An error occurs if the object is already part of the drawing layer behind text or if a drawing object is not selected.

_____

See also

Drawing Statements and Functions

DrawBringForward

DrawBringInFrontOfText

DrawBringToFront

DrawSendBackward

DrawSendToBack

DrawSendToBack

DrawSendToBack

Moves the selected drawing object behind all previous ones in a stack of drawing objects. **DrawSendToBack** does not move a drawing object from the layer in front of text to the layer behind text. An error occurs if a drawing object is not selected.

_____

See also

Drawing Statements and Functions

DrawBringForward

DrawBringInFrontOfText

DrawBringToFront

DrawSendBackward

DrawSendBehindText

DrawSetCalloutTextbox

DrawSetCalloutTextbox *TwoDimensionalArray*[**$**]*()* [**,** *Object*]

Applies the position and size stored in a two-dimensional array to the text area of the specified callout drawing object.

Argument	Explanation
TwoDimensionalArray[**$**]*()*	A two-dimensional array containing coordinates for the position and size of the text area relative to the bounding rectangle of a callout drawing object. You can use a numeric array containing values in the default measurement unit or a string array containing text measurements. For information on how values are stored in the array, see DrawGetCalloutTextbox.
Object	Specifies a drawing object: Omitted — The selected drawing object > 0 (zero) — An object whose anchor is in a range set by the **DrawSetRange** statement, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs. If the specified drawing object is not a callout, an error occurs.

For an example, see DrawCallout Example.

See also

Drawing Statements and Functions

DrawCallout

DrawGetCalloutTextbox

DrawSetInsertToTextbox

DrawSetInsertToAnchor

DrawSetInsertToAnchor [*Object*]

Moves the insertion point to the beginning of the paragraph to which the specified drawing object is anchored.

Argument	Explanation
Object	Specifies a drawing object: Omitted — The selected drawing object > 0 (zero) — An object whose anchor is in a range set by the **DrawSetRange** statement, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs.

See also

Drawing Statements and Functions

DrawSetInsertToTextBox

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawSetInsertToTextbox Example

This example sets the drawing range to the entire document and then selects the first drawing object. If the drawing object is a text box or callout, Word moves the insertion point to the text area and inserts some text.

```
DrawSetRange "\Doc"
Select Case DrawGetType(1)
    Case 3                          'Text box
        DrawSetInsertToTextbox 1
        Insert "Text box text"
    Case 8                          'Callout
        DrawSetInsertToTextbox 1
        Insert "Callout text."
    Case Else
End Select
```

DrawSetInsertToTextbox

¬Example

DrawSetInsertToTextbox [*Object*]

Moves the insertion point to the text area of the specified text box or callout drawing object. If the specified object is not a text box or callout drawing object, an error occurs.

Argument	Explanation
Object	Specifies a drawing object: Omitted — The selected drawing object > 0 (zero) — An object whose anchor is in a range set by the **DrawSetRange** statement, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs.

See also

Drawing Statements and Functions

DrawCallout

DrawSelect

DrawSetInsertToAnchor

DrawTextBox

DrawSetPolyPoints

DrawSetPolyPoints *NumPoints*, *TwoDimensionalArray*[**$**]*()* [**,** *Object*]

Applies the coordinates stored in a two-dimensional array to the specified freeform drawing object. If the specified object is not a freeform shape, an error occurs.

Argument	Explanation
NumPoints	The number of coordinates in *TwoDimensionalArray*[**$**]*()* to apply to the freeform drawing object. *NumPoints* cannot exceed the size of the array.
TwoDimensionalArray[**$**]*()*	A two-dimensional array containing the coordinates of the end points in the freeform drawing object. You can use a numeric array containing values in the default measurement unit or a string array containing text measurements. For information on how values are stored in the array, see DrawGetPolyPoints.
Object	Specifies a drawing object: Omitted — The selected drawing object > 0 (zero) — An object whose anchor is in a range set by **DrawSetRange**, where 1 is the first object in the range, 2 is the second object, and so on. If the number is not in the range 1 to **DrawCount()**, an error occurs.

For an example, see DrawCountPolyPoints() Example.

See also

Drawing Statements and Functions

DrawCountPolyPoints()

DrawFreeformPolygon

DrawGetPolyPoints

DrawReshape

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DrawSetRange Example

This example sets the drawing range to the current page, counts the number of drawing objects within the range, and then displays a message box with the result:

```
DrawSetRange "\Page"
n = DrawCount()
MsgBox "The current page contains" + Str$(n) + " drawing objects."
```

DrawSetRange, DrawSetRange()

¬Example

DrawSetRange *Bookmark$*

DrawSetRange(*Bookmark$***)**

The **DrawSetRange** statement sets the drawing range to the bookmark specified by *Bookmark$*. A drawing range establishes the group of drawing objects that other drawing statements and functions operate on. For example, the **DrawCount()** function counts the number of drawing objects whose anchors are within the drawing range.

The **DrawSetRange()** function behaves the same as the statement and also returns -1 if the range was set or 0 (zero) if the range was not set (for example, if the specified bookmark does not exist).

It is often convenient to use predefined bookmarks when setting a drawing range. For example, you can set the drawing range to the entire document by using the instruction DrawSetRange "\Doc". For more information on predefined bookmarks, see Predefined Bookmarks.

See also

Drawing Statements and Functions

DrawClearRange

DrawCount()

DrawSelect

DrawSnapToGrid

DrawSnapToGrid .SnapToGrid = *number* [**, .XGrid =** *number or text*] [**, .YGrid =** *number or text*] [**, .XOrigin =** *number or text*] [**, .YOrigin =** *number or text*]

Establishes a grid that restricts where drawing objects may be positioned and how they may be sized. A grid is useful when you want to align and connect drawing objects. The arguments for **DrawSnapToGrid** correspond to the options in the Snap To Grid dialog box (Snap To Grid button, Drawing toolbar).

Argument	Explanation
.SnapToGrid	If 1, activates the restriction on positioning drawing objects.
.XGrid	The distance, in points or a text measurement, between vertical gridlines (in other words, the increments in which drawing objects can be positioned horizontally).
.YGrid	The distance, in points or a text measurement, between horizontal gridlines (in other words, the increments in which drawing objects can be positioned vertically).
.XOrigin	The distance, in points or a text measurement, from the left edge of the page to the vertical gridline you want Word to use when laying out all other vertical gridlines. Specifying 0 (zero) or a multiple of the .XGrid measurement has no effect.
.YOrigin	The distance, in points or a text measurement, from the top edge of the page to the horizontal gridline you want Word to use when laying out all other horizontal gridlines. Specifying 0 (zero) or a multiple of the .YGrid measurement has no effect.

See also

Drawing Statements and Functions

DrawAlign

DrawTextBox

DrawTextBox

Adds a default bounded area of text to the drawing layer in front of the current text layer. A default text box is one-inch square and is inserted at the upper-left corner of the current page.

_____

See also

Drawing Statements and Functions

DrawCallout

DrawRectangle

DrawSetInsertToAnchor

DrawSetInsertToTextBox

FormatDrawingObject

DrawUngroup

DrawUngroup

Removes the association that exists between drawing objects previously associated with the **DrawGroup** statement, so you can move and size each object independently. If a group of drawing objects is not selected, an error occurs.

_____

See also

Drawing Statements and Functions

DrawExtendSelect

DrawGroup

DrawSelect

DrawUnselect

DrawUnselect

Deselects the selected drawing object and moves the insertion point to the beginning of the paragraph containing the corresponding anchor. If more than one drawing object is selected, **DrawUnselect** moves the insertion point to the first paragraph containing an anchor for at least one of the selected objects.

_____

See also

Drawing Statements and Functions

DrawExtendSelect

DrawSelect

DrawSelectNext

DrawSelectPrevious

FormatCallout

FormatCallout [**.Type** = *number*] [**, .Gap** = *number or text*] [**, .Angle** = *number*] [**, .Drop** = *text*] [**, .Length** = *number or text*] [**, .Border** = *number*] [**, .AutoAttach** = *number*] [**, .Accent** = *number*]

Sets options for callout drawing objects. The arguments for the **FormatCallout** statement correspond to the options in the Format Callout dialog box (Format Callout button, Drawing toolbar).

Argument	Explanation
.Type	Specifies the type of callout: 0 (zero) One line segment (vertical or horizontal) 1 One line segment (vertical, horizontal, or diagonal) 2 Two line segments 3 Three line segments
.Gap	Specifies the distance between the callout line and the rectangle bounding the text area, in points or a text measurement.
.Angle	Specifies the angle for the callout line (affects the outermost line segment if .Type is set to 2 or 3): 0 (zero) Any; Word adjusts the angle automatically as you move and size the callout. 1 30 degrees 2 45 degrees 3 60 degrees 4 90 degrees
.Drop	Specifies the starting position of the callout line with respect to the rectangle bounding the text area: Top Top of the text area (or if .Type is set to 1 and the line segment is horizontal, the right edge of the text area). Center Center of the text area. Bottom Bottom of the text area (or if .Type is set to 1 and the line segment is horizontal, the left edge of the text area). You can also specify a positive text measurement for the distance between the top of the text area and the starting point of the callout line.
.Length	If .Type is set to 2 or 3, the length of the first segment in the callout line in twips (20 twips = 1 point) or a text measurement, or "Best Fit" if you want Word to adjust the length automatically as you size and position the callout.
.Border	If 1, applies a border around the

	callout text.
.AutoAttach	If 1, changes the starting position of the callout line automatically when the callout origin or the callout text is switched from left to right.
.Accent	If 1, adds a vertical line next to the text area.

For an example, see DrawCallout Example.

See also

Drawing Statements and Functions

DrawCallout

DrawGetCalloutTextBox

DrawSetCalloutTextBox

FormatDrawingObject

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatDrawingObject Example

This example moves the selected drawing object down 10 points and to the right 10 points, assuming that points is the current unit of measurement:

```
Dim dlg As FormatDrawingObject
GetCurValues dlg
dlg.HorizontalPos = Val(dlg.HorizontalPos) + 10
dlg.VerticalPos = Val(dlg.VerticalPos) + 10
FormatDrawingObject dlg
```

FormatDrawingObject

¬Example

FormatDrawingObject [**.Tab** = *number*] [**, .FillColor =** *number or text*] [**, .LineColor =** *number or text*] [**, .FillPatternColor =** *number or text*] [**, .FillPattern =** *number or text*] [**, .LineType =** *number*] [**, .LineStyle =** *number*] [**, .LineWeight =** *number or text*] [**, .ArrowStyle =** *number*] [**, .ArrowWidth =** *number*] [**, .ArrowLength =** *number*] [**, .Shadow =** *number*] [**, .RoundCorners =** *number*] [**, .HorizontalPos =** *number or text*] [**, .HorizontalFrom =** *number*] [**, .VerticalPos =** *number or text*] [**, .VerticalFrom =** *number*] [**, .LockAnchor** = *number*] [**, .Height =** *number or text*] [**, .Width =** *number or text*] [**, .InternalMargin =** *number or text*]

Changes the fill color, line style, size, and position of the selected drawing object or objects. The arguments for the **FormatDrawingObject** statement correspond to the options in the Drawing Object dialog box (Format menu).

Argument	Explanation
.Tab	Specifies the tab to select when you display the Drawing Object dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) Fill tab 1 Line tab 2 Size And Position tab
.FillColor	Specifies a fill color. To specify a percentage of gray, double the percentage of gray you want and make it negative (for example, for 5 percent gray, specify --10; for 37.5 percent gray, specify --75). To specify a color, use a value between 1 and 16, as described in CharColor, where 0 (zero) indicates no color.
.LineColor	Specifies a line color. For values, see .FillColor.
.FillPatternColor	Specifies the pattern color. For values, see .FillColor.
.FillPattern	Specifies the fill pattern: 0 (zero) Sets both .FillColor and .FillPattern to "None." 1 through 25 Specifies an item in the Patterns box on the Fill tab: 1 corresponds to the first item ("None"), 2 to the second, and so on.
.LineType	Specifies whether to show the line: 0 (zero) Hides the line; only the fill and the arrowheads (if any) show. 1 Shows the line defined by .LineColor, .LineStyle, and .LineWeight.
.LineStyle	Specifies a line style in the Style box on the Line tab, where 0 (zero) is the first item, 1 is the second item, and so on through 4.
.LineWeight	The line width, in points or a text

	measurement.
.ArrowStyle	Specifies the arrow style for a linear drawing object: 0 (zero) corresponds to the first item (no arrowheads) in the Style box under Arrow Head, 1 to the second item, and so on through 6.
.ArrowWidth	Specifies the width of the arrow head for a linear drawing object: 0 (zero) Narrow 1 Medium 2 Thick
.ArrowLength	Specifies the length of the arrowhead for a linear drawing object: 0 (zero) Short 1 Medium 2 Long
.Shadow	If 1, applies a shadow effect to the drawing object.
.RoundCorners	If 1, rounds the corners of a rectangular drawing object.
.HorizontalPos	The distance, in points or a text measurement, between the reference point specified by .HorizontalFrom and the drawing object.
.HorizontalFrom	Specifies the reference point from which the horizontal position of the drawing object is measured: 0 (zero) Margin 1 Page 2 Column
.VerticalPos	The distance, in points or a text measurement, between the reference point specified by .VerticalFrom and the drawing object.
.VerticalFrom	Specifies the reference point from which the vertical position of the drawing object is measured: 0 (zero) Margin 1 Page 2 Paragraph
.LockAnchor	If 1, the drawing-object anchor remains fixed when the associated drawing object is repositioned. A locked anchor cannot be repositioned.
.Height	The height of the drawing object, in points or a text measurement.

.Width	The width of the drawing object, in points or a text measurement.
.InternalMargin	A measurement, in twips (20 twips = 1 point) or a text measurement, for the internal margin within a text box or callout drawing object.

See also

Drawing Statements and Functions

DrawAlign

DrawReshape

DrawSnapToGrid

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatPicture Example

This example resizes the selected graphic to 75 percent of the original width and height:

```
FormatPicture .ScaleX = "75%", .ScaleY = "75%"
```

FormatPicture

¬Example

FormatPicture [**.SetSize =** *number*] [**, .CropLeft =** *number or text*] [**, .CropRight =** *number or text*] [**, .CropTop =** *number or text*] [**, .CropBottom =** *number or text*] [**, .ScaleX =** *number or text*] [**, .ScaleY =** *number or text*] [**, .SizeX =** *number or text*] [**, .SizeY =** *number or text*]

Applies formatting to the selected graphic. If the current selection is not a graphic or contains a mixture of text and graphics, an error occurs. The arguments for the **FormatPicture** statement correspond to the options in the Picture dialog box (Format menu).

Argument	Explanation
.SetSize	Indicates what arguments are used to determine the size of the graphic when both the scale and size are specified: 0 (zero) .ScaleX and .ScaleY are used to format the graphic. 1 .SizeX and .SizeY are used to format the graphic.
.CropLeft, .CropRight, .CropTop, .CropBottom	Amount to crop the graphic, in points (72 points = 1 inch) or a text measurement. If you specify a negative value, the graphic is not cropped. Instead, the amount of white space around the graphic is increased.
.ScaleX, .ScaleY	The horizontal and vertical proportions of the graphic, as percentages of the original width and height.
.SizeX, .SizeY	The horizontal and vertical dimensions of the graphic, in points or a text measurement.

See also

Drawing Statements and Functions

InsertPicture

InsertDrawing

InsertDrawing

If Microsoft Draw is installed on your computer, starts Microsoft Draw and displays a blank window. When the user exits Microsoft Draw, the drawing is embedded in the active document. The instruction InsertDrawing is the same as the instruction InsertObject .Class = "MSDraw".

See also

Drawing Statements and Functions

DrawInsertWordPicture

InsertChart

InsertEquation

InsertExcelTable

InsertObject

InsertPicture

InsertSound

InsertWordArt

SelectDrawingObjects

SelectDrawingObjects

Toggles the mouse pointer shape between the standard mouse pointer and the pointer for selecting drawing objects. You drag a dotted rectangle to enclose the drawing objects you want to select.

See also

Drawing Statements and Functions

DrawExtendSelect

DrawSelect

ToggleMainTextLayer

ToggleMainTextLayer

Toggles the display of the main text layer when headers and footers are displayed. If the selection is not within a header or footer, an error occurs.

See also

Section and Document Formatting

ViewHeader

ToggleScribbleMode

ToggleScribbleMode

Toggles hand annotation mode on and off. When you activate hand annotation mode, Word switches the active document to page layout view. This statement is for use with Windows for Pen Computing only.

ViewDrawingToolbar

ViewDrawingToolbar

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

See also

Drawing Statements and Functions

ViewBorderToolbar

ViewToolbars

AutoMarkIndexEntries

AutoMarkIndexEntries *ConcordanceFilename$*

Automatically indexes the active document using *ConcordanceFilename$*, the path and filename of a concordance file. A concordance file is a Word document containing a two-column table with terms to index in the first column and index entries in the second column. The **AutoMarkIndexEntries** statement inserts an XE (Index Entry) field with the appropriate entry text after each occurrence of the terms listed in the first column of the concordance file. For more information on automatic indexing, see Chapter 20, "Indexes, Tables of Contents, and Other Tables," in the *Microsoft Word User's Guide*.

See also

Editing Statements and Functions

MarkIndexEntry

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Cancel Example

This example pastes the contents of the Clipboard into a document, selects the pasted text, and then places the text of the selection into a variable. To accomplish this task, the macro inserts a bookmark at the insertion point, pastes the text, and then, using **ExtendSelection**, selects the pasted text. The **Cancel** statement turns off extend mode, so that when the last instruction moves the insertion point to the end of the document, no text is selected.

```
CopyBookmark "\Sel", "temp"
EditPaste
ExtendSelection
EditGoTo "temp"
pasted$ = Selection$()
Cancel
EndOfDocument
```

Cancel

¬Example

Cancel

Cancels a mode activated by the **ColumnSelect**, **CopyFormat**, **CopyText**, or **ExtendSelection** statements.

See also

Editing Statements and Functions

OK

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ChangeCase Example

This example selects the current paragraph (using the predefined bookmark "\Para") and capitalizes the first letter of each sentence:

```
EditGoTo "\Para"
ChangeCase 4
```

ChangeCase, ChangeCase()

¬Example

ChangeCase [*Type*]

ChangeCase()

The **ChangeCase** statement sets the case of the selected text to sentence case, lowercase, uppercase, or initial capital letters. **ChangeCase** does not change the character formats associated with the selected text, as do the **SmallCaps** and **AllCaps** statements. **ChangeCase** corresponds to the Change Case command (Format menu).

Argument	Explanation
Type	Specifies the change in case:
	Omitted — If the selection is one sentence or less, alternates the case of the selection among all lowercase, all uppercase, and initial capital letters. If the selection is more than one sentence, alternates among all lowercase, all uppercase, and sentence case.
	0 (zero) — Sets the text to all lowercase
	1 — Sets the text to all uppercase
	2 — Capitalizes the first letter of each selected word
	3 — Capitalizes the first letter of the selection
	4 — Capitalizes the first letter of each selected sentence
	5 — Toggles the case of each selected letter (for example, "Word" becomes "wORD")

If there is no selection, Word selects the word nearest the insertion point, and then changes the case of the selected word.

The **ChangeCase()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selected text is in uppercase
1	If all of the selected text is in uppercase
2	If the text is in a mixture of uppercase and lowercase

See also

Editing Statements and Functions

AllCaps

FormatChangeCase

LCase$()

SmallCaps

UCase$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CopyText Example

This example inserts a copy of the current paragraph (selected using the predefined bookmark "\Para") immediately below the current paragraph:

```
EditGoTo "\Para"
CopyText
ParaDown
OK
```

CopyText

¬Example

CopyText

Copies the selected text without putting it on the Clipboard (same as pressing SHIFT+F2). For **CopyText** to work, macro instructions must make a selection immediately before the **CopyText** instruction, make a new selection immediately after **CopyText**, and then use the **OK** statement to copy the text.

See also

Editing Statements and Functions

Cancel

MoveText

OK

DeleteBackWord

DeleteBackWord

Deletes the word immediately preceding the insertion point or the selection, but does not place it on the Clipboard.

If there is a selection, **DeleteBackWord** behaves as if there were an insertion point; the statement does not delete the selection but deletes the word immediately preceding it. If the insertion point is in the middle of a word, **DeleteBackWord** deletes the characters between the insertion point and the start of the word.

Note that if there is a space character between the word and the insertion point, **DeleteBackWord** deletes the space character as well as the word. Also, if the insertion point follows a punctuation mark, such as a comma or period, **DeleteBackWord** deletes the punctuation mark only.

_____

See also

Editing Statements and Functions

DeleteWord

EditClear

EditCut

WordLeft

DeleteWord

DeleteWord

Deletes the word immediately following the insertion point or the first word or part of a word included in the selection, but does not place it on the Clipboard. If the insertion point is in the middle of a word, **DeleteWord** deletes the characters following the insertion point to the end of the word.

Note that if there is a space character between the insertion point and the word, **DeleteWord** deletes the space character as well as the word. Also, if the insertion point precedes a punctuation mark, such as a comma or period, **DeleteWord** deletes the punctuation mark only.

_____

See also

Editing Statements and Functions

DeleteBackWord

EditClear

EditCut

WordRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditClear Example

This example deletes five characters to the right of the insertion point or, if there is a selection, deletes the selection plus four characters:

```
EditClear 5
```

EditClear

¬Example

EditClear [*Count*]

Deletes the selection or a specified number of characters. Unlike **EditCut**, **EditClear** does not change the contents of the Clipboard.

Argument	Explanation
Count	Specifies the number and location of characters to delete: >0 (zero) Deletes characters to the right of the insertion point. Omitted Deletes the selection or the character to the right of the insertion point. <0 (zero) Deletes characters to the left of the insertion point. Note that EditClear 0 has no effect.

Note

If *Count* is nonzero and there is a selection, Word deletes the selection and counts it as one character against *Count*.

See also

Editing Statements and Functions

EditCut

EditCopy

EditCopy

Copies the selection to the Clipboard. If there is no selection, an error occurs.

_____

See also

Editing Statements and Functions

EditCut

EditPaste

EditPasteSpecial

EditCut

EditCut

Removes the selection from the document and places it on the Clipboard. If there is no selection, an error occurs.

_____

See also

Editing Statements and Functions

EditCopy

EditPaste

EditPasteSpecial

Spike

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditGoTo Example

This example counts the number of tables in the document. The bookmark "Temp" is defined at each successive insertion point location so **CmpBookmarks()** can determine whether the most recent **EditGoTo** (or **RepeatFind**) instruction has moved the insertion point. When **RepeatFind** no longer moves the insertion point --- that is, when there are no more tables --- Word exits the **While...Wend** loop.

```
StartOfDocument
EditBookmark "Temp", .Add
count = 0
If SelInfo(12) = - 1 Then count = 1
EditGoTo .Destination = "t"
While CmpBookmarks("\Sel", "Temp") <> 0
    EditBookmark "Temp", .Add
    count = count + 1
    RepeatFind
Wend
EditBookmark "Temp", .Delete
MsgBox "There are" + Str$(count) + " tables in the document"
```

EditGoTo

¬Example

EditGoTo .Destination = *text*

Moves the insertion point or selection to the specified location or item, such as a page, bookmark, footnote, line, or field. To move the insertion point to a destination listed in the following table, use the corresponding identifier in combination with a number, the plus sign (+) or minus sign (-), or a string.

Destination	Identifier	Example text for .Destination
Page	p (or omitted)	The text "p5" or "5" goes to the fifth page.
Section	s	The text "s" goes to the next section.
Line	l	The text "l+5" goes to the fifth line after the current insertion point. The plus sign (+) means count forward from the insertion point.
Bookmark	(none)	The text "Temp" goes to the bookmark "Temp."
Annotation	a	The text "a'Sara Levine'" goes to the next mark for an annotation by Sara Levine.
Footnote	f	The text "f-" goes to the previous footnote reference mark. The minus sign (-) means count backward from the insertion point.
Endnote	e	The text "e5" goes to the fifth endnote reference mark.
Field	d	The text "d'TIME'" goes to the next TIME field. The single quotation marks separate the field type from the identifier.
Table	t	The text "t" goes to the next table.
Graphic	g	The text "g10" goes to the tenth graphic in the document.
Equation	q	The text "q-" goes to the previous equation.
Object	o	The text "o-'WordArt'" goes to the previous Microsoft WordArt object. The single quotation marks separate the object type from the identifier.

The following table summarizes the use of numbers and the plus sign or minus sign with destination identifiers in **EditGoTo** instructions. Numbers and the plus sign or minus sign can be used with every type of destination except bookmarks.

Identifier and this	Result
Number	Goes to an item according to its location within the document. For example, EditGoto "l30" goes to the

	thirtieth line in a document; EditGoto "f200" goes to the two-hundredth footnote.
+[*Number*] or -[*Number*]	Goes to an item relative to the current location. For example, EditGoto "l+2" goes to the second line after the current line. Likewise, EditGoto "l-2" goes to the second line before the current line. The instruction EditGoto "l-" goes to the previous line; EditGoto "l+" goes to the next line.
Omitted	Goes to the next item specified. For example, EditGoto "s" goes to the next section in the document.

See also

Editing Statements and Functions

EditFind

GoBack

GoToNext*Item*

GoToPrevious*Item*

NextField

NextObject

NextPage

PrevField

PrevObject

PrevPage

RepeatFind

EditPaste

EditPaste

Inserts the contents of the Clipboard at the insertion point. **EditPaste** replaces the selection if you select the Typing Replaces Selection check box on the Edit tab in the Options dialog box (Tools menu).

_____

See also

Editing Statements and Functions

CopyText

EditCopy

EditCut

EditPasteSpecial

MoveText

EditRedo

EditRedo

Redoes the last action that was undone (reverses **EditUndo**).

_____

See also

Editing Statements and Functions

EditRepeat

EditUndo

EditRepeat

EditRepeat

Repeats the most recent editing action, if possible.

_____

See also

Editing Statements and Functions

EditRedo

EditUndo

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditTOACategory Example

This example changes the ninth item in the list of categories to "Case Histories":

```
EditTOACategory .Category = 8, .CategoryName = "Case Histories"
```

EditTOACategory

¬Example

EditTOACategory .Category = *number*, **.CategoryName =** *text*

Modifies the name of a category of citations for a table of authorities. The arguments for the **EditTOACategory** statement correspond to the options in the Edit Category dialog box (Mark Citation dialog box, Index And Tables command, Insert menu).

Argument	Explanation
.Category	The category you want to modify: 1 corresponds to the first name in the list, 2 to the second name, and so on.
.CategoryName	The new name for the category of citations.

See also

Editing Statements and Functions

InsertTableOfAuthorities

MarkCitation

EditUndo

EditUndo

Undoes the last action, if possible. Most editing and formatting actions can be undone. Some actions, such as modifying a style, cannot be undone.

_____

See also

Editing Statements and Functions

EditRedo

EditRepeat

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ExtendMode() Example

This example toggles the state of extend mode:

```
If ExtendMode() = 0 Then ExtendSelection Else Cancel
```

ExtendMode()

¬Example

ExtendMode()

Returns -1 if extend mode is on. In extend mode, actions that normally move the insertion point extend the selection. **ExtendMode()** does not indicate whether column selection mode is on.

_____

See also

Editing Statements and Functions

Cancel

ColumnSelect

ExtendSelection

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertAddCaption Example

This example adds "Diagram" to the list of available caption labels:

```
InsertAddCaption .Name = "Diagram"
```

InsertAddCaption

¬Example

InsertAddCaption [**.Name =** *text*]

Adds a new item to the Label box in the Caption dialog box (Insert menu). The new item becomes a valid string for the .Label argument in an **InsertCaption** instruction.

Argument	Explanation
.Name	The name for the new caption label

See also

Editing Statements and Functions

InsertAutoCaption

InsertCaption

InsertCaptionNumbering

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertAutoCaption Example

This example creates a new caption label, "Picture," and specifies that whenever a Microsoft Paintbrush object is inserted, this label is automatically inserted:

```
InsertAddCaption .Name = "Picture"
InsertAutoCaption .Object = "Paintbrush Picture", \
        .Label = "Picture"
```

InsertAutoCaption

¬Example

InsertAutoCaption [**.Clear**] [**, .ClearAll**] [**, .Object =** *text*] [**, .Label =** *text*] [**, .Position =** *number*]

Specifies a caption to insert automatically whenever an object of the specified type is inserted. The arguments for the **InsertAutoCaption** statement correspond to the options in the AutoCaption dialog box (Caption command, Insert menu).

Argument	Explanation
.Clear	Cancels adding captions automatically for the type of object specified by .Object.
.ClearAll	Cancels adding captions automatically for all types of objects.
.Object	The name of the object type for which you want to establish an automatic caption, as it appears in the Add Caption When Inserting box.
.Label	The caption label to insert automatically. If the label has not yet been defined, an error occurs. Use **InsertAddCaption** to define new labels.
.Position	The position of the caption: 0 (zero) Above Item 1 Below Item

See also

Editing Statements and Functions

InsertAddCaption

InsertCaption

InsertCaptionNumbering

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertBreak Example

This example inserts a page break before each Heading 1 paragraph in the active document:

```
StartOfDocument
EditFindClearFormatting
EditFindStyle .Style = "Heading 1"
EditFind .Find = "", .Direction = 0, .Wrap = 0
While EditFindFound()
    CharLeft
    InsertBreak .Type = 0
    EditFind .Find = "", .Direction = 0, .Wrap = 0
Wend
```

InsertBreak

¬Example

InsertBreak [**.Type =** *number*]

Inserts a page, column, or section break at the insertion point. The values available for the .Type argument correspond to the options in the Break dialog box (Insert menu).

Argument	Explanation
.Type	The type of break to insert: 0 (zero) or omitted Page break 1 Column break 2 Next Page section break 3 Continuous section break 4 Even Page section break 5 Odd Page section break 6 Line break (newline character)

See also

Editing Statements and Functions

InsertColumnBreak

InsertPageBreak

InsertSectionBreak

ParaPageBreakBefore

TableSplit

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertCaption Example

This example inserts a caption with the label "Figure" and the title ": WordArt Object" after each WordArt object in the active document. **GetSelStartPos()** returns the character position of the insertion point before and after the **EditGoTo** instruction. When the character positions are the same, indicating there are no more WordArt objects, the **While...Wend** loop is exited.

```
StartOfDocument
cp1 = GetSelStartPos()
EditGoTo "o'WordArt'"
cp2 = GetSelStartPos()
While cp1 <> cp2
    CharRight 1, 1
    InsertCaption .Label = "Figure", \
                  .Title = ": WordArt Object", .Position = 1
    cp1 = GetSelStartPos()
    EditGoTo "o'WordArt'"
    cp2 = GetSelStartPos()
Wend
```

InsertCaption

¬Example

InsertCaption [**.Label =** *text*] [**, .TitleAutoText =** *text*] [**, .Title =** *text*] [**, .Delete**] [**, .Position =** *number*]

Inserts a caption above or below a selected item. The arguments for the **InsertCaption** statement correspond to the options in the Caption dialog box (Insert menu).

Argument	Explanation
.Label	The caption label to insert. If the label has not yet been defined, an error occurs. Use **InsertAddCaption** to define new labels.
.TitleAutoText	The AutoText entry whose contents you want to insert after the label in the caption (overrides any text specified for .Title).
.Title	The text to insert after the label in the caption (ignored if .TitleAutoText is specified).
.Delete	Removes the specified Label from the label box. Built-in labels (such as "Figure") cannot be deleted; if .Label specifies a built-in label, an error occurs.
.Position	When an item is selected, specifies whether to insert the caption above or below the item: 0 (zero) Above Selected Item 1 Below Selected Item

See also

Editing Statements and Functions

InsertAddCaption

InsertAutoCaption

InsertCaptionNumbering

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertCaptionNumbering Example

This example defines a number format for captions containing the "Figure" label. Chapter numbers are excluded from the sequence number.

```
InsertCaptionNumbering .Label = "Figure", .FormatNumber = 3, \
        .ChapterNumber = 0
```

InsertCaptionNumbering

¬Example

InsertCaptionNumbering [**.Label =** *text*] [**, .FormatNumber =** *number*] [**, .ChapterNumber =** *number*] [**, .Level =** *number*] [**, .Separator =** *text*]

Defines a format for sequence numbers in captions for a specific type of item. The arguments for the **InsertCaptionNumbering** statement correspond to the options in the Caption Numbering dialog box (Caption command, Insert menu).

Argument	Explanation
.Label	The label for which you want to define a sequence number format. If the label has not yet been defined, an error occurs. Use **InsertAddCaption** to define new labels.
.FormatNumber	Specifies a default format for the sequence number in captions containing the specified label: 0 (zero) 1, 2, 3, ... 1 a, b, c, ... 2 A, B, C, ... 3 i, ii, iii, ... 4 I, II, III, ...
.ChapterNumber	If 1, a chapter number is included by default in captions containing the specified label. Chapters are determined by the heading style specified by .Level.
.Level	A number corresponding to the heading style used for chapter headings in the active document: 1 corresponds to Heading 1, 2 corresponds to Heading 2, and so on.
.Separator	The character to use as a separator between the chapter number and the sequence number.

See also

Editing Statements and Functions

InsertAddCaption

InsertAutoCaption

InsertCaption

InsertColumnBreak

InsertColumnBreak

Inserts a column break at the insertion point. If the insertion point is in a table, **InsertColumnBreak** splits the table by inserting a paragraph mark above the row containing the insertion point. The instruction InsertColumnBreak is the same as the instruction InsertBreak .Type = 1.

_____

See also

Editing Statements and Functions

InsertBreak

InsertPageBreak

InsertSectionBreak

TableSplit

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertCrossReference Example

This example inserts a sentence containing two cross-references to a diagram: one to the caption and another to the page containing the diagram. A typical sentence inserted with these instructions might be, "For a diagram of a widget, see 'Figure 1: Standard Widget' on page 5."

```
Insert "For a diagram of a widget, see " + Chr$(34)
InsertCrossReference .ReferenceType = "Figure", \
        .ReferenceKind = "2", .ReferenceItem = "1"
Insert Chr$(34) + " on page "
InsertCrossReference .ReferenceType = "Figure", \
        .ReferenceKind = "7", .ReferenceItem = "1"
Insert "."
```

InsertCrossReference

¬Example

InsertCrossReference [**.ReferenceType =** *text*] [**, .ReferenceKind =** *text*] [**, .ReferenceItem =** *text*]

Inserts a cross-reference to a heading, bookmark, footnote, endnote, or an item for which a caption label is defined (for example, an equation, figure, or table). The arguments for the **InsertCrossReference** statement correspond to the options in the Cross-reference dialog box (Insert menu).

Argument	Explanation
.ReferenceType	The type of item to which you want to insert a cross-reference, as it appears in the Reference Type box.
.ReferenceKind	A number specified as text corresponding to the information you want the cross-reference to include. See the following table for available values.
.ReferenceItem	A number specified as text corresponding to an item in the For Which [Reference Type] box: "1" corresponds to the first item, "2" corresponds to the second item, and so on.

The following table shows the values available for .ReferenceKind based on the setting for .ReferenceType.

.ReferenceType	Values available for .ReferenceKind	
Heading	0 (zero)	Heading Text
	7	Page Number
	8	Heading Number
Bookmark	1	Bookmark Text
	7	Page Number
	9	Paragraph Number
Footnote	5	Footnote Number
	7	Page Number
Endnote	6	Endnote Number
	7	Page Number
An item for which a caption label is defined	2	Entire Caption
	3	Only Label And Number
	4	Only Caption Text
	7	Page Number

See also

Editing Statements and Functions

InsertCaption

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertIndex Example

This example inserts a simple, indented index that replaces the current index, if one exists:

```
InsertIndex .Type = 0, .HeadingSeparator = 0, .Replace = 1
```

InsertIndex

¬Example

InsertIndex [**.HeadingSeparator =** *number*] [**, .Replace =** *number*] [**, .Type =** *number*] [**, .RightAlignPageNumbers =** *number*] [**, .Columns =** *number*]

Compiles and inserts an index at the insertion point or replaces an existing index. The index is the result of an INDEX field that you can update when index entries change.

Argument	Explanation
.HeadingSeparator	Specifies the heading separator: 0 (zero) or omitted None 1 Blank Line 2 Letter
.Replace	Specifies whether to replace the index, if it already exists: 0 (zero) or omitted The existing index is not replaced (the instruction has no effect other than to select the index). 1 The existing index is replaced.
.Type	Specifies the type of index: 0 (zero) or omitted Indented 1 Run-in
.RightAlignPageNumbers	If 1, aligns page numbers with the right edge of the column.
.Columns	The number of columns you want in the index.

See also

Editing Statements and Functions

AutoMarkIndexEntries

MarkIndexEntry

InsertPageBreak

InsertPageBreak

Inserts a page break at the insertion point. The instruction InsertPageBreak is the same as the instruction InsertBreak .Type = 0.

See also

Editing Statements and Functions

InsertBreak

InsertColumnBreak

InsertSectionBreak

ParaPageBreakBefore

TableSplit

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertPageNumbers Example

This example adds a right-aligned page number to the header and specifies that the page number not appear on the first page:

```
InsertPageNumbers .Type = 0, .Position = 2, .FirstPage = 0
```

InsertPageNumbers

¬Example

InsertPageNumbers [**.Type =** *number*] [**, .Position =** *number*] [**, .FirstPage =** *number*]

Inserts a PAGE field inside a frame in the header or footer and positions the frame as specified. The arguments for the **InsertPageNumbers** statement correspond to the options in the Page Numbers dialog box (Insert menu).

Argument	Explanation
.Type	Specifies where to add the page field: 0 (zero) Header 1 Footer
.Position	Specifies the position of the framed PAGE field: 0 (zero) Left 1 Center 2 Right 3 Inside (left on odd pages and right on even pages) 4 Outside (right on odd pages and left on even pages)
.FirstPage	If 1, the field is included in the header or footer on the first page.

See also

Editing Statements and Functions

FormatPageNumber

InsertPageField

ViewHeader

InsertSpike

InsertSpike

Empties the special AutoText entry "Spike" and inserts its contents at the insertion point. For an example, see Spike Example.

_____

See also

Editing Statements and Functions

EditAutoText

InsertAutoText

Spike

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertSymbol Example

This example inserts a double-headed arrow at the insertion point:

```
InsertSymbol .Font = "Symbol", .CharNum = "171"
```

InsertSymbol

¬Example

InsertSymbol .Font = *text*, **.Tab =** *number*, **.CharNum =** *number or text*

Inserts a symbol at the insertion point. Note that **InsertSymbol** does not insert a SYMBOL field as in Word for Windows version 2.*x*. The arguments for the **InsertSymbol** statement correspond to the options in the Symbol dialog box (Insert menu).

Argument	Explanation
.Font	The name of the font containing the symbol. Names of decorative fonts appear in the Font box in the Symbol dialog box.
.Tab	Specifies which tab to select when you display the Symbol dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) Symbol tab 1 Special Characters tab
.CharNum	The sum of 31 and the number corresponding to the position of the symbol on the table of symbols, counting from left to right. For example, to specify an omega (), at position 56 on the table of symbols in the Symbol font, set .CharNum to 87.

See also

Editing Statements and Functions

Chr$()

InsertTableOfAuthorities

InsertTableOfAuthorities [**.Replace =** *number*] [**, .Passim =** *number*] [**, .KeepFormatting =** *number*] [**, .Category =** *number*]

Collects citations from TA (Table of Authorities Entry) fields and inserts a TOA (Table of Authorities) field at the insertion point.

Argument	Explanation
.Replace	Specifies whether to replace a previously compiled table of authorities: 0 (zero) or omitted — The existing table of authorities is not replaced (the instruction has no effect other than selecting the field result). 1 — The existing table of authorities is replaced.
.Passim	If 1, Word replaces five or more different page references to the same authority with "passim."
.KeepFormatting	If 1, Word retains the formatting of long citations.
.Category	A number corresponding to an item in the Category box that specifies the type of citations to collect: 0 (zero) corresponds to the first item ("All"), 1 corresponds to the second item, and so on.

See also

Editing Statements and Functions

InsertTableOfContents

InsertTableOfFigures

MarkCitation

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertTableOfContents Examples

This example inserts a table of contents using Heading 1 through Heading 3 styles. TC fields, if any, are excluded.

```
InsertTableOfContents .Outline = 1, .From = 1, .To = 3, \
        .Fields = 0, .RightAlignPageNumbers = 1, .Replace = 1
```

The following example inserts a table of contents using Heading 1 through Heading 3 styles and Heading 5 styles (Heading 4 styles are skipped). Note that headings formatted with the Heading 5 style are formatted with the TOC 4 style in the table of contents.

```
InsertTableOfContents .Outline = 1, .From = 1, .To = 3, \
    .AddedStyles = "Heading 5,4", .RightAlignPageNumbers = 1, \
    .Replace = 0
```

The following example inserts a table of contents based on text formatted with the "Intro," "Main heading," "Summary," and "Subhead" styles. Text formatted with the Intro, Main heading, and Summary styles is mapped to the TOC 1 style; text formatted with the Subhead style is mapped to the TOC 2 style.

```
custom$ = "Intro,1,Summary,1,Main heading,1,Subhead,2"
InsertTableOfContents .AddedStyles = custom$, \
        .RightAlignPageNumbers = 1, .Replace = 0
```

InsertTableOfContents

¬Example

InsertTableOfContents [**.Outline =** *number*] [**, .Fields =** *number*] [**, .From =** *number*] [**, .To =** *number*] [**, .TableId =** *text*] [**, .AddedStyles =** *text*] [**, .Replace =** *number*] [**, .RightAlignPageNumbers =** *number*]

Collects table of contents entries from headings, TC (Table of Contents Entry) fields, or both, and inserts a TOC (Table of Contents) field at the insertion point.

Argument	Explanation
.Outline	If 1, collects table of contents entries from heading styles.
.Fields	If 1, collects table of contents entries from TC (Table of Contents Entry) fields.
.From	If .Outline is 1, the highest level of heading style to collect.
.To	If .Outline is 1, the lowest level of heading style to collect.
.TableId	The entry identifier specified in those TC fields you want to include in the table of contents.
.AddedStyles	Styles to collect in addition to the heading styles indicated by .From and .To. Use the following syntax when specifying .AddedStyles: *StyleToCollect1*,*TOC#*[,*StyleToCollect2*,*TOC#*][,...] You can specify any number of styles to collect, though the length of the string value cannot be longer than 256 characters. For example, .AddedStyles = "NoteStyle,3" causes text formatted with a style called "NoteStyle" to be collected into a table of contents and formatted with the TOC 3 style.
.Replace	Specifies whether to replace a previously compiled table of contents: 0 (zero) or omitted — The existing table of contents is not replaced (the instruction has no effect other than to select the table). 1 — The existing table of contents is replaced.
.RightAlignPageNumbers	If 1, aligns page numbers with the right margin.

See also

Editing Statements and Functions

InsertTableOfAuthorities

InsertTableOfFigures
MarkTableOfContentsEntry

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertTableOfFigures Example

This example inserts a table of figures marked by captions with the label "Diagram." If a table of diagrams already exists, it is replaced.

```
InsertTableOfFigures .Caption = "Diagram", .Label = 1, \
        .RightAlignPageNumbers = 1, .Replace = 1
```

InsertTableOfFigures

¬Example

InsertTableOfFigures [**.Caption =** *text*] [**, .Label =** *number*] [**, .RightAlignPageNumbers =** *number*] [**, .Replace =** *number*]

Collects captions with the specified label and inserts them at the insertion point.

Argument	Explanation
.Caption	The label that identifies the items for which you want to create a table.
.Label	If 1, includes labels and sequence numbers with captions in the table of figures.
.RightAlignPageNumbers	If 1, aligns page numbers with the right margin.
.Replace	Specifies whether to replace a previously compiled table of figures: 0 (zero) or omitted — The existing table of figures is not replaced (the instruction has no effect other than to select the table). 1 — The existing table of figures is replaced.

See also

Editing Statements and Functions

InsertAutoCaption

InsertCaption

InsertTableOfAuthorities

InsertTableOfContents

MarkCitation

MarkCitation [**.LongCitation =** *text*] [**, .LongCitationAutoText =** *text*] [**, .Category** = *number*] [**, .ShortCitation =** *text*] [**, .NextCitation**] [**, .Mark**] [**, .MarkAll**]

Inserts a TA (Table of Authorities Entry) field next to the text you select or next to every instance of the selected text. For more information, see TA (Table of Authorities Entry) field.

The arguments for the **MarkCitation** statement correspond to the options in the Mark Citation dialog box (Table Of Authorities tab, Index And Tables command, Insert menu).

Argument	Explanation
.LongCitation	The long citation for the entry as it will appear in the table of authorities.
.LongCitationAutoText	The name of an AutoText entry containing the long citation text you want to appear in the table of authorities (.LongCitation is ignored).
.Category	The number of the category (in the order of the list of categories in the Mark Citation dialog box) to associate with the entry: 1 corresponds to the first category in the list, 2 to the second category, and so on.
.ShortCitation	The short citation for the entry as it will appear in the list under Short Citation in the Mark Citation dialog box.
.NextCitation	Finds the next instance of the text specified by .ShortCitation.
.Mark	Inserts a TA (Table of Authorities Entry) field to the right of the selected text.
.MarkAll	Inserts a TA (Table of Authorities Entry) field to the right of all instances of the selected text.

If you don't specify .NextCitation, .Mark, or .MarkAll, Word inserts a TA field to the right of the selected text.

See also

Editing Statements and Functions

InsertTableOfAuthorities

MarkIndexEntry

MarkTableOfContentsEntry

MarkIndexEntry

MarkIndexEntry [**.MarkAll**] [**, .Entry =** *text*] [**, .Range =** *text*] [**, .Bold =** *number*] [**, .Italic =** *number*] [**, .CrossReference =** *text*] [**, .EntryAutoText =** *text*] [**, .CrossReferenceAutoText =** *text*]

Inserts an XE (Index Entry) field next to the text you select or next to every instance of the selected text. For more information, see XE (Index Entry) field.

The arguments for the **MarkIndexEntry** statement correspond to the options in the Mark Index Entry dialog box (Index tab, Index And Tables command, Insert menu).

Argument	Explanation
.MarkAll	Inserts an XE field after each instance of the selected text.
.Entry	The text you want to appear in the index in the form *MainEntry*[:*Subentry*].
.Range	The name of a bookmark marking the range of pages you want to appear in the index. If you don't specify .Range, the number of the page containing the XE field appears in the index.
.Bold	If 1, page numbers for the entry are bold in the index.
.Italic	If 1, page numbers for the entry are italic in the index.
.CrossReference	A cross-reference that will appear in the index.
.EntryAutoText	The name of an AutoText entry containing the text you want to appear in the index (.Entry is ignored).
.CrossReferenceAutoText	The name of an AutoText entry containing the text for a cross-reference (.CrossReference is ignored).

See also

Editing Statements and Functions

InsertIndex

MarkCitation

MarkTableOfContentsEntry

MarkTableOfContentsEntry

MarkTableOfContentsEntry [**.Entry =** *text*] [**, .EntryAutoText =** *text*] [**, .TableId =** *text*] [**, .Level =** *text*]

Inserts a TC (Table of Contents Entry) field next to the selected text. For more information, see TC (Table of Contents Entry) field.

Argument	Explanation
.Entry	The text you want to appear in the table of contents
.EntryAutoText	The name of an AutoText entry containing the text you want to appear in the table of contents (.Entry is ignored)
.TableId	A one-letter identifier for the type of item (for example, "i" for an illustration)
.Level	A level for the entry in the table of contents

See also

Editing Statements and Functions

InsertTableOfContents

MarkCitation

MarkIndexEntry

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MoveText Example

This example moves the line containing the insertion point to the beginning of the document:

```
StartOfLine
EndOfLine 1
MoveText
StartOfDocument
OK
```

MoveText

¬Example

MoveText

Moves text without changing the contents of the Clipboard (the same as pressing the F2 key). For **MoveText** to work, macro instructions must make a selection immediately before the **MoveText** instruction, make a new selection immediately after **MoveText**, and then use the **OK** statement to move the text.

See also

Editing Statements and Functions

CopyFormat

CopyText

OK

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OK Example

This example copies the line containing the insertion point to the beginning of the document without changing the contents of the Clipboard:

```
StartOfLine
EndOfLine 1
CopyText
StartOfDocument
OK
```

OK

¬Example

OK

Completes a **CopyText** or **MoveText** operation (the same as pressing ENTER while the operation is in progress).

See also

Editing Statements and Functions

Cancel

CopyText

MoveText

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Overtype Example

If overtype mode is active, this example displays a message box with Yes and No buttons asking if overtype should be deactivated. If the user chooses Yes, which returns a value of -1, overtype is deactivated.

```
If Overtype() = -1 Then
    Button = MsgBox("Overtype is on. Turn off?", 4)
    If button = -1 Then Overtype 0
End If
```

Overtype, Overtype()

¬Example

Overtype [*On*]

Overtype()

The **Overtype** statement switches between overtype and insert modes. In overtype mode, characters you type replace existing characters one by one; in insert mode, characters you type move existing text to the right.

Argument	Explanation
On	Specifies the mode: 0 (zero) Switches to insert mode. 1 Switches to overtype mode ("OVR" appears in status bar). Omitted Switches between overtype and insert modes.

The **Overtype()** function returns the following values.

Value	Explanation
0 (zero)	If overtype mode is off
-1	If overtype mode is on

Note that text you insert with the **Insert** statement will not overwrite text in the document, regardless of whether the document is in insert or overtype mode.

See also

Editing Statements and Functions

ToolsOptionsGeneral

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Spike Example

This example opens RESEARCH.DOC, uses **Spike** to collect all the Heading 1 paragraphs, closes RESEARCH.DOC without saving changes, and then inserts the headings into the active document:

```
FileOpen .Name = "C:\DOCS\RESEARCH.DOC", .ReadOnly = 1
EditFindClearFormatting
EditFindStyle .Style = "Heading 1"
EditFind .Find = "", .Direction = 0, .Format = 1
While EditFindFound()
    Spike
    ParaDown
    EditFind .Find = "", .Direction = 0, .Format = 1
Wend
FileClose 2                    'Close without saving
InsertSpike                    'Empty the Spike at insertion point
```

Spike

¬Example

Spike

Deletes the current selection and adds it to a built-in AutoText entry called the Spike. Items added to the Spike are separated by paragraph marks.

You can use the Spike to collect text and graphics from various locations in one or more documents and then insert them all together using **InsertSpike**, which clears the Spike. To insert the Spike contents without clearing the Spike, you use **EditAutoText**.

_____

See also

Editing Statements and Functions

EditAutoText

EditCut

InsertSpike

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Beep Examples

This example produces a beep immediately before displaying an input box:

```
Beep
name$ = InputBox$("The macro has finished running. " + \
                                        "Save document as:")
```

The following example makes the computer beep three times. The second **For...Next** loop nested within the first one provides a delay between beeps. In effect, the delay loop makes Word count to a thousand before producing the next beep. Without the delay loop, Word would produce beeps so quickly that they would sound like a single, continuous beep. You can vary the delay between beeps by increasing or decreasing the end value of the second **For...Next** loop. For example, changing the end value to 5000 would increase the delay between beeps; changing it to 100 would decrease the delay.

```
Sub BeepThreeTimes
    For x = 1 to 3
        Beep
      For timer = 1 To 1000                    'Delay loop between beeps
        Next timer
    Next x
End Sub
```

Beep

¬Example

Beep [*BeepType*]

Causes the computer's speaker to beep. A typical use of **Beep** is to signal the end of a long process or to indicate that an error has occurred.

Argument	Explanation
BeepType	The type of beep: 1 (or omitted), 2, 3, or 4. On most computers, all numbers produce the same sound --- the sound you hear when an error occurs.

Note

A beep will not sound if the [Windows] section of WIN.INI includes the line Beep = No.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DOSToWin$() Example

This example opens a sequential file created by an MS-DOS application, translates each line to the Windows character set, and places the result in a new sequential file:

```
ChDir "C:\TMP"
Open "DOS.TXT" For Input As #1
Open "WINDOWS.TXT" For Output As #2
While Not Eof(1)
    Line Input #1, temp$
    Print #2, DOSToWin$(temp$)
Wend
Close
```

DOSToWin$()

¬Example

DOSToWin$(***StringToTranslate$*)**

Translates a string from the original equipment manufacturer (OEM) character set to the Windows character set.

The OEM character set is typically used by MS-DOS applications. Characters 32 through 127 are usually the same in the OEM and Windows character sets. The other characters in the OEM character set (0 (zero) through 31 and 128 through 255) are generally different from the Windows characters.

See also

Environment Statements and Functions

WinToDOS$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Environ$() Example

This example checks the PATH environment variable and displays one of two message boxes based on the number of characters in the path. (The limit is 127 characters.)

```
path$ = Environ$("PATH")
length = Len(path$)
avail = 127 - length
Select Case avail
    Case 127
        MsgBox "The PATH is empty."
    Case 0 To 126
        MsgBox "You can add" + Str$(avail) + \
                        " character(s) to the PATH."
    Case Else
End Select
```

Environ$()

¬Example

Environ$(***EnvironmentVariable$*)**

Returns a string associated with an MS-DOS *EnvironmentVariable$*. For example, tempdir$ = Environ$("TEMP") returns the directory associated with the TEMP environment variable in MS-DOS.

Environment variables are normally set using MS-DOS batch files (such as AUTOEXEC.BAT). For more information about environment variables, see your MS-DOS documentation.

See also

Environment Statements and Functions

AppInfo$()

GetProfileString$()

GetSystemInfo

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Err Example

This macro prompts the user to type a style name that is then applied to the selected paragraphs. The macro contains an error handler that illustrates the use of **Err** within a **Select Case** control structure. The **Select Case** control structure in this macro can be described as follows:

- If the error code is 102 (which occurs if the **InputBox$** dialog box is canceled), the macro prevents the "Command failed" error message from being displayed.
- If the error code is 24 (which occurs if the style name does not exist), Word displays a custom message box in place of the "Bad parameter" error message, error trapping is reset with the instruction `Err = 0`, and the **InputBox$** dialog box is displayed again so the user can type another style name. Note that `Case 24` is the only case in which error trapping needs to be reset. In the other cases, the macro ends and there is no need for further error trapping.
- If the error code is neither 102 nor 24, the instruction `Error Err` displays the error.

```
Sub MAIN
On Error Goto trap
start:
apply$ = InputBox$("Type a style name:", "Apply Style")
Style apply$
trap:
Select Case Err
    Case 102                        'Input box was canceled
    Case 24                         'Bad parameter
        MsgBox "No such style; try another name."
        Err = 0
        Goto start
    Case Else                       'Any other error condition
        Error Err
End Select
End Sub
```

Err

¬Example

Err = 0

Error Err

ErrorNum **= Err**

Err is a special variable whose value is the error code for the most recent error condition.

- **Err = 0** is used to reset error trapping after an error has occurred, normally at the end of an error trap.
- **Error Err** displays the message associated with the most recent error condition and stops the macro.
- *ErrorNum* **= Err** assigns the value of **Err** to the variable *ErrorNum*.

For more information on trapping errors, see On Error.

_____

See also

Environment Statements and Functions

Error

Goto

On Error

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Error Example

This example generates WordBasic error condition 53 in order to test the handling of this error. When error condition 53 occurs, Word displays a custom message box; any other error runs the instruction Error Err, which displays the built-in message associated with the error number.

```
On Error Goto trap
Error 53                                'Generate error condition
trap:
If Err = 53 Then
    MsgBox "Error 53 successfully trapped"
    Err = 0
Else
    Error Err
End If
```

Error

¬Example

Error Err

Error *ErrorNumber*

Error Err generates the most recent error condition and stops the macro. **Error** *ErrorNumber* generates the error condition associated with *ErrorNumber.* For WordBasic error conditions, which have numbers of less than 1000, **Error** displays an error message. For Word errors, which have numbers of 1000 or greater, **Error** generates an error condition, but does not display an error message. **Error** can be used to test an error trap.

For more information on error trapping, see On Error and Err. For a list of errors and their numbers, see Error Messages.

See also

Environment Statements and Functions

Err

Goto

On Error

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetPrivateProfileString$() Example

This example sets the variable artdir$ to the path specified by the PICTURE-PATH key in the [Microsoft Word] section of WINWORD6.INI and, if the setting exists, changes the current directory accordingly. The last two instructions display the Insert Picture dialog box (Picture command, Insert menu).

```
artdir$ = GetPrivateProfileString$("Microsoft Word", \
    "PICTURE-PATH", "C:\WIN31\WINWORD6.INI")
If artdir$ <> "" Then ChDir artdir$
Dim dlg As InsertPicture
x = Dialog(dlg)
```

For another example, see SetPrivateProfileString Example.

GetPrivateProfileString$()

¬Example

GetPrivateProfileString$(***Section$*,** *KeyName$***,** *Filename$***)**

Returns a setting in a settings file. A settings file is a text file such as WIN.INI that your macros can use to store and retrieve settings. For example, you can store the name of the active document when you quit Word so that it can be reopened automatically the next time you start Word.

Argument	Explanation
Section$	The name of the section in the settings file that contains *KeyName$*. The section name appears between brackets before the associated keys (do not include the brackets with *Section$*).
KeyName$	The key whose setting you want to retrieve. In the settings file, the key name is followed by an equal sign (=) and the setting.
Filename$	The path and filename for the settings file.

If a specified section, key, or file does not exist, **GetPrivateProfileString$()** returns an empty string.

See also

Environment Statements and Functions

GetProfileString$()

SetPrivateProfileString

SetProfileString

GetProfileString$()

GetProfileString$(**[*Section$*,**] *KeyName$***)**

Returns a setting in WIN.INI, or if *Section$* is "Microsoft Word 2.0," "Microsoft Word," "MSWord Text Converters," or "MSWord Editable Sections," returns the setting from WINWORD6.INI instead. These exceptions are made for compatibility with Word version 2.*x* macros.

Argument	Explanation
Section$	The name of the section in the settings file that contains *KeyName$*. The section name appears between brackets before the associated keys (do not include the brackets with *Section$*). If omitted, "Microsoft Word" is assumed.
KeyName$	The key whose setting you want to retrieve. In the settings file, the key name is followed by an equal sign (=) and the setting.

If a specified section or key does not exist, **GetProfileString$()** returns an empty string ("").

See also

Environment Statements and Functions

GetPrivateProfileString$()

SetPrivateProfileString

SetProfileString

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

IsDocumentDirty() Example

This example saves the active document if there are unsaved changes:

```
If IsDocumentDirty() = -1 Then FileSave
```

IsDocumentDirty()

¬Example

IsDocumentDirty()

Returns a value indicating whether the active document has changed since it was last saved (that is, if it is "dirty"). Whether a document is dirty determines if Word displays a prompt to save changes when the document is closed.

Value	Explanation
0 (zero)	If the document has not changed since it was last saved
-1	If the document has changed since it was last saved

See also

Environment Statements and Functions

IsTemplateDirty()

SetDocumentDirty

SetTemplateDirty

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

IsExecuteOnly() Example

This example determines whether the global macro Test is execute-only. If it isn't, Word displays the macro in the macro-editing window.

```
encrypted = IsExecuteOnly("NORMAL.DOT:Test")
If encrypted = 0 Then ToolsMacro .Name = "Test", .Show = 1, .Edit
```

IsExecuteOnly()

¬Example

IsExecuteOnly([*Macro$*]**)**

Returns 0 (zero) if the specified macro can be edited or -1 if the macro is execute-only. Macros that cannot be edited are sometimes referred to as *encrypted*. You can make a macro execute-only using **MacroCopy**.

Argument	Explanation
Macro$	The name of a macro. Use the following syntax: [*TemplateName***:**]*MacroName$* If omitted, the active template and active macro are assumed. The template specified by *TemplateName* must be open in a document window, attached to an open document, or loaded as a global template; otherwise **IsExecuteOnly()** generates an error.

See also

Environment Statements and Functions

IsMacro()

MacroCopy

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

IsTemplateDirty() Example

This example saves the active template if there are unsaved changes:

```
If IsTemplateDirty() = -1 Then SaveTemplate
```

IsTemplateDirty()

¬Example

IsTemplateDirty()

Returns a value indicating whether the active template has changed since it was last saved (that is, if it is "dirty"). Whether a template is dirty determines if Word displays a prompt to save changes when the template is closed.

Value	Explanation
0 (zero)	If the template has not changed since it was last saved
-1	If the template has changed since it was last saved

See also

Environment Statements and Functions

IsDocumentDirty

SaveTemplate

SetDocumentDirty()

SetTemplateDirty

ScreenRefresh

ScreenRefresh

Updates the display on your monitor to show the current screen display. You can use **ScreenRefresh** in a macro after a **ScreenUpdating** instruction has turned off screen updating. **ScreenRefresh** turns on screen updating for just one instruction and then immediately turns it off. Subsequent instructions do not update the screen until screen updating is turned on again with another **ScreenUpdating** instruction. You can use **ScreenRefresh** and **ScreenUpdating** to create a series of screen "snapshots." During a long process, the screen can be updated from time to time to indicate the macro's progress.

For an example, see ScreenUpdating Example.

See also

Environment Statements and Functions

ScreenUpdating

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ScreenUpdating Example

This example arranges document windows in three different ways. A **ScreenUpdating** statement is used to turn off screen updating while the windows are being arranged. When the windows are arranged, a **ScreenRefresh** statement displays them. Note that the instructions TileDocWindowsVertically and CascadeDocWindows call subroutines (not shown here) that tile and cascade document windows.

```
ScreenUpdating 0
TileDocWindowsVertically
ScreenRefresh
MsgBox "Windows are tiled vertically."
CascadeDocWindows
ScreenRefresh
MsgBox "Windows are cascaded."
WindowArrangeAll
ScreenUpdating 1
MsgBox "Windows are tiled horizontally."
```

ScreenUpdating, ScreenUpdating()

¬Example

ScreenUpdating [*On*]

ScreenUpdating()

The **ScreenUpdating** statement controls most display changes on the monitor while a macro is running. When screen updating is turned off, Word still allows the macro to display or retrieve information using status bar prompts, input boxes, dialog boxes, or message boxes. You can increase the speed of some macros by preventing screen updates. Screen updating is restored when the macro finishes or when it stops after an error.

Argument	Explanation
On	Specifies whether to show screen updates: 0 (zero) The screen does not update while the macro is running. 1 The screen updates normally while the macro is running. Omitted Toggles screen updating.

The **ScreenUpdating()** function returns the following values.

Value	Explanation
0 (zero)	Screen updating is turned off.
1	Screen updating is turned on.

See also

Environment Statements and Functions

ScreenRefresh

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelInfo() Examples

This example determines whether the current selection extends over a page break, either manual or automatic. Note that SelInfo(3) is used twice: once to return the page number of the end of the selection, and again to return the page number of the beginning of the selection. The macro is then able to compare these values to determine how many page breaks are in the selection.

```
ToolsRepaginate
EditBookmark .Name = "Tmp"
endPage = SelInfo(3)
SelType 1
startPage = SelInfo(3)
EditGoTo .Destination = "Tmp"
numBreaks = endPage - startPage
If numBreaks = 1 Then
    MsgBox "Selection extends over one page break."
ElseIf numBreaks > 1 Then
    MsgBox "Selection extends over" + Str$(numBreaks) + \
        " page breaks."
End If
EditBookmark .Name = "Tmp", .Delete
```

The following example increases the view magnification by 10 percent:

```
n = SelInfo(19)
If n < 190 Then
    ViewZoom .ZoomPercent = n + 10
ElseIf n = 200 Then
    MsgBox "Already zoomed to maximum."
Else
    ViewZoom .ZoomPercent = 200
End If
```

SelInfo()

¬Example

SelInfo(*Type***)**

Returns one of 36 types of information about the selection, where *Type* is one of the values described in the following tables. In the explanations, the active end of the selection refers to the end that moves when you press SHIFT+an arrow key.

Type	Explanation
1	Number of the page containing the active end of the selection. If you set a starting page number or make other manual adjustments, returns the adjusted page number (unlike SelInfo(3)). If the selection is in a header or footer pane in normal view, returns -1. If the selection is in a footnote or annotation pane, returns the page number of the first footnote or annotation in the selection.
2	Number of the section containing the active end of the selection.
3	Number of the page containing the active end of the selection, counting from the beginning of the document. Any manual adjustments to page numbering are disregarded (unlike SelInfo(1)).
4	Number of pages in the document.

Type 5 is valid only in page layout view. Type 6 is valid in page layout view or in normal view if you select the Background Repagination check box on the General tab and clear the Draft Font check box on the View tab in the Options dialog box (Tools menu).

Type	Explanation
5	Horizontal position of the selection; distance between the left edge of the selection and the left edge of the page, in twips (20 twips = 1 point, 72 points = 1 inch). Returns -1 if the selection is not visible.
6	Vertical position of the selection; distance between the top edge of the selection and the top edge of the page, in twips (20 twips = 1 point, 72 points = 1 inch). Returns -1 if the selection is not visible in the document window.

Types 7 and 8 are most useful in page layout view, where you can display text boundaries by selecting the Text

Boundaries check box on the View tab in the Options dialog box (Tools menu). Text boundaries include table cells, frames, the edges of the page, text columns, and so on. For a complete list of text boundaries, see NextObject.

Type	Explanation
7	Horizontal position of the selection, relative to the left edge of the nearest text boundary enclosing it, in twips (20 twips = 1 point, 72 points = 1 inch). Returns -1 if selection is not visible.
8	Vertical position of the selection, relative to the upper edge of the nearest text boundary enclosing it, in twips (20 twips = 1 point, 72 points = 1 inch). Especially useful for determining the position of the insertion point within a frame or table cell. For a complete list of text boundaries, see NextObject. Returns -1 if the selection is not visible.
9	Character position of the first character in the selection or, if no text is selected, the character to the right of the insertion point (same as the character column number displayed in the status bar after "Col").
10	Line number of the first character in the selection; if Background Repagination is cleared or if Draft Font is selected, returns -1.
11	Returns -1 if the selection is an entire frame.
12	Returns -1 if the selection is within a table.

Types 13 through 18 apply only if the selection is within a table; if the selection is not within a table, the function returns -1.

Type	Explanation
13	The row number containing the beginning of the selection.
14	The row number containing the end of the selection.
15	The number of rows in the table.
16	The table column number containing the beginning of the selection.
17	The table column number containing the end of the selection.

18	The greatest number of columns within any row in the selection.

Types 19 through 36 return miscellaneous information.

Argument	Explanation
19	The current percentage of magnification as set by **ViewZoom**.
20	The current selection mode: returns 0 (zero) for normal selection, 1 for extended selection, and 2 for column selection. Corresponds to the box in the status bar that reads either "EXT" or "COL."
21	Returns -1 if Caps Lock is in effect.
22	Returns -1 if Num Lock is in effect.
23	Returns -1 if Word is in overtype mode.
24	Returns -1 if revision marking is in effect.
25	Returns -1 if the selection is in the footnote or endnote pane, or in a footnote or endnote in page layout view. See also values 35 and 36.
26	Returns -1 if the selection is in an annotation pane.
27	Returns -1 if the selection is in a macro-editing window
28	Returns -1 if the selection is in the header or footer pane or in a header or footer in page layout view.
29	The number of the bookmark enclosing the start of the selection; 0 (zero) if none or invalid. The number corresponds to the position of the bookmark in the document --- 1 for the first bookmark, 2 for the second, and so on.
30	The number of the last bookmark that starts before or at the same place as the selection; returns 0 (zero) if none or invalid.
31	Returns -1 if the insertion point is at the end-of-row mark in a table.
32	Returns one of the following values, depending on where the

	selection is in relation to a footnote, endnote, or annotation reference: -1 If the selection includes but is not limited to a footnote, endnote, or annotation reference 0 (zero) If the selection is not in a footnote, endnote, or annotation reference 1 If the selection is in a footnote reference 2 If the selection is in an endnote reference 3 If the selection is in an annotation reference
33	The kind of header or footer containing the selection: -1 None (the selection is not in a header or footer) 0 (zero) Even-page header 1 Odd-page header (or the only header if there are not odd and even headers) 2 Even-page footer 3 Odd-page footer (or the only footer if there are not odd and even footers) 4 First-page header 5 First-page footer
34	Returns -1 if the active document is a master document (that is, if it contains at least one subdocument).
35	Returns -1 if the selection is in the footnote pane or in a footnote in page layout view.
36	Same as 35, but for endnotes.

See also

Environment Statements and Functions

Selection$()

SelType

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelType Example

This example selects the next occurrence of the word "background" and then cancels the selection, leaving the insertion point at the beginning of the word:

```
EditFind .Find = "background", .Direction = 0
If EditFindFound() Then SelType 1
```

SelType, SelType()

¬Example

SelType *Type*

SelType()

The **SelType** statement specifies how the insertion point or selected text is indicated in your document. For example, you can specify that selected text appear with a dotted underline instead of in reverse video, which may be preferable on some video displays.

SelType is more commonly used to position the insertion point at the beginning of the current selection. For this result, **SelType** has advantages over **CharLeft** because if there is no current selection, the insertion point does not move left.

Argument	Explanation
Type	The type of highlight: 1 Solid insertion point (default) 2 Solid selection (default) 4 Dotted selection or insertion point (whichever is current) Note that the **SelType()** function never returns this value because 5 and 6 provide more specific information. 5 Dotted insertion point (visible in **CopyText** and **MoveText** modes) 6 Dotted selection (visible in **CopyText** and **MoveText** modes)

The **SelType()** function returns a number corresponding to the type of highlight.

See also

Environment Statements and Functions

Selection$()

SelInfo()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetDocumentDirty Example

This example assumes that a main document for a mail merge is active. The instructions print form letters to a document, which is then made clean. When the document is closed, Word will not display a prompt or save changes.

```
MailMergeToDoc
SetDocumentDirty 0
```

SetDocumentDirty

¬Example

SetDocumentDirty [*Dirty*]

Controls whether Word recognizes a document as "dirty" --- that is, changed since the last time the document was saved. When a dirty document is closed, Word displays a prompt asking if changes should be saved.

If you change a document and then set *Dirty* to 0 (zero), Word recognizes the document as unchanged, or "clean." When you close the document, Word neither displays a prompt nor saves changes.

Argument	Explanation
Dirty	Specifies whether to make the active document dirty: 0 (zero) The document is treated as clean. 1 or omitted The document is recognized as dirty.

See also

Environment Statements and Functions

IsDocumentDirty()

IsTemplateDirty()

SetTemplateDirty

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetPrivateProfileString Example

This pair of auto macros automatically opens the document that was active the last time you quit Word.

The following is an AutoExit macro:

```
Sub MAIN
a$ = FileName$()
check = SetPrivateProfileString("Word Info", "LastActive", a$, \
    "MY.INI")
If check = 0 Then MsgBox "Could not set INI option."
End Sub
```

The following is an AutoExec macro:

```
Sub MAIN
name$ = GetPrivateProfileString$("Word Info", "LastActive", \
    "MY.INI")
FileOpen .Name = name$
End Sub
```

For information on auto macros such as AutoExit and AutoExec, see Auto Macros.

SetPrivateProfileString, SetPrivateProfileString()

¬Example

SetPrivateProfileString *Section$*, *KeyName$*, *Setting$*, *Filename$*

SetPrivateProfileString(*Section$*, *KeyName$*, *Setting$*, *Filename$***)**

Defines or redefines a setting in a settings file. A settings file is a text file like WIN.INI that your macros can use for storing and retrieving settings. For example, you can store the name of the active document when you quit Word so that it can be reopened automatically the next time you start Word.

Argument	Explanation
Section$	The name of the section in the settings file that contains the option you want to set. This is the name that appears between brackets before the associated keys (do not include the brackets with *Section$*).
KeyName$	The key to set. In the settings file, the key name is followed by an equal sign (=) and the setting.
Setting$	The new setting.
Filename$	The filename for the settings file. If a path is not specified, the Windows directory is assumed. If the file does not already exist, Word creates it.

The **SetPrivateProfileString()** function behaves the same as the statement and also returns a value indicating whether the action was successful: -1 means the key was set, 0 (zero) means the key was not set. Keys cannot be set if the settings file is read-only.

See also

Environment Statements and Functions

GetPrivateProfileString$()

GetProfileString$()

SetProfileString

ToolsAdvancedSettings

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetProfileString Example

This example uses **GetSystemInfo$()** to determine the "sCountry" setting in the [intl] section of WIN.INI; **SetProfileString** changes the key's setting to "Sweden" if it does not already have that setting.

```
If GetSystemInfo$(29) <> "Sweden" Then
    SetProfileString "intl", "sCountry", "Sweden"
End If
```

SetProfileString

¬Example

SetProfileString *Section$, KeyName$, Setting$*

Creates a key and defines a setting for it, or redefines the setting of an existing key in WIN.INI. If *Section$* is "Microsoft Word 2.0," "Microsoft Word," "MSWord Text Converters," or "MSWord Editable Sections," **SetProfileString** sets the key in WINWORD6.INI instead. These exceptions are made for compatibility with Word version 2.*x* macros.

Argument	Explanation
Section$	The name of the section in WIN.INI that contains the key you want to set. This is the name that appears between brackets before the associated keys (do not include the brackets with *Section$*).
KeyName$	The key to set. In WIN.INI, the key name is followed by an equal sign (=) and the setting.
Setting$	The new setting.

See also

Environment Statements and Functions

GetPrivateProfileString$()

GetProfileString$()

SetPrivateProfileString

ToolsAdvancedSettings

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetTemplateDirty Example

This example makes a temporary change in the active template's keyboard assignments (assigns the **SmallCaps** command to CTRL+SHIFT+C) and then makes the template clean. When the template is closed, Word does not display a prompt or save changes.

```
ToolsCustomizeKeyboard .KeyCode = 835, .Context = 0, \
    .Name = "SmallCaps", .Add
SetTemplateDirty 0
```

SetTemplateDirty

¬Example

SetTemplateDirty [*Dirty*]

Controls whether Word recognizes a template as "dirty"---that is, changed since the last time the template was saved. When a dirty template is closed, Word displays a prompt asking if changes should be saved.

If you change a template and then set *Dirty* to 0 (zero), Word recognizes the template as unchanged, or "clean." When you close the template, Word neither displays a prompt nor saves changes.

Argument	Explanation
Dirty	Specifies whether to make the active template dirty: 0 (zero) The template is treated as clean. 1 or omitted The template is recognized as dirty.

See also

Environment Statements and Functions

IsDocumentDirty()

IsTemplateDirty()

SetDocumentDirty

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

WaitCursor Example

This example suppresses the hourglass pointer for the first half of the total iterations of a **For...Next** loop and then displays the hourglass for the second half:

```
WaitCursor 0
For i = 1 To 1000
    If i = 500 Then WaitCursor 1
Next i
```

WaitCursor

¬Example

WaitCursor *Wait*

Changes the mouse pointer from the current pointer to an hourglass, or vice versa. Control of the pointer is restored to Word when the macro ends.

Argument	Explanation
Wait	Specifies the mouse pointer to display: 0 (zero) The current pointer 1 The hourglass pointer

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

WinToDOS$() Example

This example opens a sequential file created by a Windows-based application, translates each line to the OEM character set, and places the result in a new sequential file:

```
ChDir "C:\TMP"
Open "WINDOWS.TXT" For Input As #1
Open "DOS.TXT" For Output As #2
While Not Eof(1)
    Line Input #1, temp$
    Print #2, WinToDOS$(temp$)
Wend
Close
```

WinToDOS$()

¬Example

WinToDOS$(***StringToTranslate$*)**

Translates a string from the Windows character set to the original equipment manufacturer (OEM) character set.

The OEM character set is typically used by MS-DOS applications. Characters 32 through 127 are usually the same in the OEM and Windows character sets. The other characters in the OEM character set (0 through 31 and 128 through 255) are generally different from the Windows characters.

See also

Environment Statements and Functions

DOSToWin$()

DoFieldClick

DoFieldClick

Moves the insertion point to the location specified by the selected GOTOBUTTON field, or runs the macro specified by the selected MACROBUTTON field (the same as double-clicking with the left mouse button or pressing ALT+SHIFT+F9 within the result of the field).

For more information, see GOTOBUTTON field and MACROBUTTON field.

GetFieldData$()

GetFieldData$()

When the insertion point is within an ADDIN field, retrieves text data stored internally within the field. Data stored within an ADDIN field is not visible, even when field codes are showing. If there is a selection, it must begin at an ADDIN field; otherwise, an error occurs.

See also

Fields Statements and Functions

PutFieldData

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertField Examples

This example inserts an AUTHOR field:

```
InsertField .Field = "AUTHOR"
```

The following example nests a FILLIN field within a COMMENTS field. The instructions display all field codes, insert a COMMENTS field, then insert the FILLIN field. The instruction SendKeys "{tab}{enter}" automatically chooses the OK button when the fill-in dialog box is displayed.

```
ToolsOptionsView .FieldCodes = 1
InsertField .Field = "COMMENTS"
CharLeft
SendKeys "{tab}{enter}"
InsertField .Field = "FILLIN " + Chr$(34) + \
        "Type your comment" + Chr$(34)
```

InsertField

¬Example

InsertField .Field = *text*

Inserts the specified field at the insertion point.

Argument	Explanation
.Field	The field type and instructions to insert. For information on a specific field, including syntax and examples, select the field name in the Field dialog box (Insert menu) and then press F1, or see the corresponding entry in Field Types.

Do not include the field characters (**{}**), but follow all other syntax rules for field codes. To insert quotation marks in field codes, use Chr$(34).

See also

Fields Statements and Functions

InsertFieldChars

InsertFieldChars

InsertFieldChars

Inserts field characters (**{}**) at the insertion point and then positions the insertion point between the field characters.

_____

See also

Fields Statements and Functions

InsertField

InsertPageField

InsertPageField

Inserts a PAGE field without any field instructions at the insertion point.

See also

Fields Statements and Functions

InsertDateField

InsertField

InsertPageNumbers

InsertTimeField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

LockFields Example

This example inserts a DATE field and then immediately locks the field. This is useful if you want Word to insert the date for you, but you don't want the date to be updated each time you print the document.

InsertDateField

PrevField

LockFields

LockFields

¬Example

LockFields

Prevents the fields within the selection from being updated.

_____

See also

Fields Statements and Functions

UnlinkFields

UnlockFields

UpdateFields

PutFieldData

PutFieldData *FieldData$*

When the insertion point is within an ADDIN field, stores the text data specified by *FieldData$* in the field. The data is stored internally in the field and is not visible, even when field codes are showing. If there is a selection, the selection must begin at an ADDIN field; otherwise, an error occurs.

_____

See also

Fields Statements and Functions

GetFieldData$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToggleFieldDisplay Example

This example uses the **ViewFieldCodes** statement to change the display of all fields to field results, and then changes the display of fields in the current paragraph (selected using the predefined bookmark "\Para") back to field codes:

```
ViewFieldCodes 0
EditGoTo .Destination = "\Para"
ToggleFieldDisplay
```

ToggleFieldDisplay

¬Example

ToggleFieldDisplay

Toggles the display of the fields in the selection between field codes and field results. If the selection does not contain at least one field, an error occurs. The exception to this rule is in page layout view, where **ToggleFieldDisplay** toggles the display of all fields.

See also

Fields Statements and Functions

ViewFieldCodes

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsManageFields Example

This example renames the "Name1" field to "FirstName":

```
ToolsManageFields .FieldName = "Name1", .NewName = "FirstName", \
        .Rename
```

ToolsManageFields

¬Example

ToolsManageFields [**.FieldName =** *text*] [**, .Add**] [**, .Remove**] [**, .Rename**] [**, .NewName =** *text*]

Adds, removes, or renames a field name in a mail-merge data source or header source. If the active document does not contain a table with the specified field in the first row of the table, an error occurs.

Argument	Explanation
.FieldName	The field name you want to add, remove, or rename
.Add	Adds the specified field name to a new table column
.Remove	Removes the specified field name and the associated table column
.Rename	Renames the specified field name
.NewName	A new name for the field; used with .Rename

See also

Fields Statements and Functions

MailMergeEditDataSource

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

UnlinkFields Example

This example unlinks all fields in the active document. First, Word displays a message box giving the user the option to update all fields before unlinking. Then, depending on the user's response, Word carries out the appropriate action.

```
answer = MsgBox("Update all fields before unlinking?", \
    "Unlink All Fields", 35)
On Error Goto trap
Select Case answer
    Case -1                         'Yes
        EditSelectAll
        UpdateFields
        UnlinkFields
    Case 0                          'No
        EditSelectAll
        UnlinkFields
    Case 1                          'Cancel
        Goto bye
trap:
    If Err = 102 Then MsgBox "No fields to unlink." Else Error Err
bye:
End Select
```

UnlinkFields

¬Example

UnlinkFields

Replaces the selected fields with their most recent results. When you unlink a field, it is converted to regular text or graphics and can no longer be updated automatically. Note that some fields cannot be unlinked --- for example, XE (Index Entry) fields, which show no result. If the selection does not contain fields that can be unlinked, an error occurs.

_____

See also

Fields Statements and Functions

LockFields

UnlockFields

UpdateFields

UnlockFields

UnlockFields

Allows selected fields that were previously locked with **LockFields** to be updated.

See also

Fields Statements and Functions

LockFields

UnlinkFields

UpdateFields

UpdateFields

UpdateFields

Updates the selected fields.

See also

Fields Statements and Functions

LockFields

UnlinkFields

UnlockFields

UpdateSource

UpdateSource

Saves the changes made to the result of an INCLUDETEXT field back to the source document. The source document must be formatted as a Word document.

See also

Fields Statements and Functions

UpdateFields

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Close Example

This example opens a file, inserts a list of AutoText entries in the Normal template and any loaded global templates, and then closes the file:

```
Open "AUTOTEXT.TXT" For Output As #1
For count = 1 To CountAutoTextEntries(0)
    Print #1, AutoTextName$(count)
Next count
Close #1
```

Close

¬Example

Close [[#]*FileNumber*]

Closes an open sequential file. *FileNumber* is the number specified in the **Open** instruction that opened the file for input, output, or appending. If *FileNumber* is omitted, all files that were opened with **Open** are closed.

Sequential files, which are opened with **Open** and closed with **Close**, are not displayed in document windows. Although you can use **Open** to open any file, **Open** and **Close** are intended to be used with text files. **Close** does not display a prompt when it closes a file. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

See also

Basic File Input/Output Statements and Functions

Eof()

Input

Input$()

Line Input

Lof()

Open

Print

Read

Seek

Write

Eof()

Eof([#]*FileNumber***)**

Determines whether the end of an open sequential file has been reached. *FileNumber* is the number specified in the **Open** instruction that opened the file for input. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

You cannot use this function to check for the end of a Word document open in a document window. To check for the end of a Word document, use the predefined bookmark "\EndOfDoc" with **CmpBookmarks()**. For more information about predefined bookmarks, see Predefined Bookmarks.

Eof() returns the following values.

Value	Explanation
-1	The end of the file specified by *FileNumber* has been reached.
0 (zero)	The end of the file has not been reached.

For an example, see Read Example.

See also

Basic File Input/Output Statements and Functions

Close

Input

Input$()

Line Input

Lof()

Open

Print

Read

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Input Examples

This example opens for input a list of files to save in rich-text format (RTF). Assume that FILES.TXT is a text file in which each paragraph contains two string values: the filename to open and the filename to save. The instructions open each file in turn and save it in RTF.

```
Open "C:\DOCS\FILES.TXT" For Input As #1
While Not Eof(#1)
    Input #1, docname$, rtfname$
    FileOpen "C:\DOCS\" + docname$
    FileSaveAs .Name =  "C:\RTF\" + rtfname$, .Format = 6
    FileClose 2
Wend
Close #1
```

The following example displays the prompt "RTF filename?" in the status bar and defines the variable rtfname$ with the text the user types:

```
Input "RTF filename", rtfname$
```

Input

¬Example

Input #*FileNumber***,** *Variable1*[**$**] [**,** *Variable2*[**$**]] [**,** *Variable3*[**$**]] [**,** ...]

Input [*Prompt$***,**] *Variable1*[**$**] [**,** *Variable2*[**$**]] [**,** *Variable3*[**$**]] [**,** ...]

Retrieves string or numeric values from a sequential file, typically containing data added with the **Print** statement, and assigns the values to variables. *FileNumber* is the number specified in the **Open** instruction that opened the file for input. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

Input is similar to **Read** in that it retrieves comma-delimited values from sequential files, but unlike **Read**, it does not remove quotation marks from string values. **Input** can accept strings with as many as 65,280 characters. Longer strings are truncated. For more information, see Read.

If *FileNumber* is omitted, the user is prompted with a question mark (?) in the status bar to type one or more values, separated by commas. The user presses ENTER to end keyboard input. If *Prompt$* is specified, the question mark follows the string.

_____

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input$()

InputBox$()

Line Input

Lof()

Open

Print

Read

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Input$() Example

This example defines the variable filecode$ with the first 10 characters in the text file INFO.TXT:

```
Open "INFO.TXT" For Input As #1
filecode$ = Input$(10, #1)
Close #1
```

Input$()

¬Example

Input$(*NumChars***,** [#]*FileNumber***)**

Reads *NumChars* characters (as many as 65,280) from an open sequential file. *FileNumber* is the number specified in the **Open** instruction that opened the file for input. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

InputBox$()

Line Input

Lof()

Open

Print

Read

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Line Input Examples

This example places one line from the file designated as #1 into the variable sample$:

```
Line Input #1, sample$
```

The following example displays the default prompt (a question mark) in the status bar. The variable key$ is defined as the text entered after the question mark.

```
Line Input key$
```

The following example displays the prompt "Search text: " in the status bar. The variable target$ is defined as the text entered after the prompt.

```
Line Input "Search text: ", target$
```

Line Input

¬Example

Line Input **#***FileNumber***,** *Variable***$**

Line Input [*Prompt$***,**] *Variable***$**

Reads an entire line from an open sequential file and places it into a string variable. In a sequential file, a line is terminated by the carriage-return (ANSI 13) and linefeed (ANSI 10) characters. *FileNumber* is the number specified by the **Open** instruction that opened the file for input. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

Line Input is similar to the **Input** and **Read** statements, but **Line Input** does not break the line into separate values at commas. It places the entire line into a single string variable. **Line Input** can accept lines as long as 65,280 characters. Longer lines are truncated. Unlike **Read**, which takes only the text within quotation marks, **Line Input** does not distinguish between quotation marks and other characters.

If **#***FileNumber* is omitted, the user is prompted with a question mark (?) in the status bar to type a value. The user presses ENTER to end keyboard input. If *Prompt$* is specified, the question mark is not displayed.

_____

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

Input$()

Lof()

Open

Print

Read

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Lof() Example

This example uses **Open** to prepare the file MYDATA.TXT for sequential input and then displays the size of the file MYDATA.TXT in the status bar:

```
Open "MYDATA.TXT" For Input As #1
    Size = Lof(1)
    Print Size; " bytes"
Close #1
```

Lof()

¬Example

Lof([#]*FileNumber***)**

Returns the length of an open sequential file, in bytes. *FileNumber* is the number specified in the **Open** instruction that opened the file for input. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

Input$()

Line Input

Open

Print

Read

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Open Example

This macro saves a rich-text format (RTF) copy of each document listed in the text file C:\DOCS\CONVERT.TXT. The **Open** instruction opens CONVERT.TXT for input as a sequential file. The instructions in the **While...Wend** loop run once for each document, assigning the filename listed next in CONVERT.TXT to curdoc$, opening the document, saving an RTF copy, and then closing the document.

```
Sub MAIN
Open "C:\DOCS\CONVERT.TXT" For Input As #1
While Not Eof(1)
    Line Input #1, curdoc$
    FileOpen .Name = "C:\DOCS\" + curdoc$
    FileSaveAs .Name = "C:\DOCS\" + curdoc$, .Format = 6
    FileClose 2
Wend
Close #1
End Sub
```

Open

¬Example

Open *Name$* **For** *Mode$* **As** [#]*FileNumber*

Opens a sequential file for input or output of text. You can use sequential files to supply values for macro variables or to store data generated by macros.

Sequential files, which are opened with **Open** and closed with **Close**, are not displayed in document windows. Although you can use **Open** to open any file, **Open** and **Close** are intended to be used with text files. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

Argument	Explanation
Name$	The name of the file to open
Mode$	The mode in which the file is opened: **Input** Opens the text file so that data may be read from the file into the macro. When you open a file for input, you can use **Read**, **Input**, **Line Input**, and **Input$()** instructions to read from the file. If *Name$* does not exist, an error occurs. **Output** Opens the text file so that data may be written from the macro to the file. When you open a file for output, you can use **Write** and **Print** instructions to write to the file. If *Name$* does not exist, Word creates it. If *Name$* already exists, Word deletes its contents. **Append** Opens the text file so that data may be written from the macro to the file. When you open a file to append, you can use **Write** and **Print** instructions to write to the file. The existing contents of the file remain and the information you write is added to the end of the file. If *Name$* does not exist, Word creates it.
FileNumber	A number to assign the file, from 1 through 4

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

Input$()

Line Input

Lof()

Print

Read

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Print Examples

This example displays the following sentence in the status bar: "March sales were 2500 and May sales were 3600 for a total of 6100."

```
mar = 2500
may = 3600
Print "March sales were"; mar ; " and May sales were" \
    ; may ; " for a total of" ; mar + may ; "."
```

The following example displays this line on the status bar: "Juan Garcia 32 01234".

```
name$ = "Juan Garcia"
age = 32
employeeNum = 01234
Print name$; age; employeeNum
```

The following example sets up a sequential file for use as a text-only data source in a mail merge operation. The first **Print** instruction inserts column headings; the second adds a data record. Items in a given **Print** instruction are inserted into a single paragraph and are separated by tabs. Note that WordBasic doesn't have a sequential-file-access instruction that can read tab-delimited data. To see the results, you must open the sequential file as you would any Word document.

```
Open "C:\TMP\DATA.TXT" For Output As #1
Print #1, "Name", "Address", "Occupation"
Print #1, "Juan Garcia", "123 Main St", "Accountant"
Close #1
```

Print

¬Example

Print [*#FileNumber,*] *Expression1*[*$*] [**;** *or* **,** *Expression2*[*$*] [**;** *or* **,** ...]

Displays the specified expressions in the status bar, or if you specify *FileNumber*, sends the expressions to the sequential file associated with that number. Unlike **MsgBox**, which accepts only strings, **Print** accepts strings, string variables, numbers, and numeric variables, and allows you to mix all these types.

You can join strings and string variables with the plus sign (+). You can join strings and numbers with a semicolon (;) or a comma (,). A comma inserts a tab between values.

Note

Print automatically adds a space before positive numeric values; there is no need to include an extra space in a string that precedes one. **Print** does not add a space before negative numeric values.

Like **Write**, the **Print** statement can send expressions to a sequential file. However, the expressions are formatted differently in the file, as illustrated in the following table. Note that **Print** separates the expressions with a tab character.

WordBasic instruction	Line created in sequential file #1
Print #1, "Phil", "Teacher"	Phil Teacher
Write #1, "Phil", "Teacher"	"Phil","Teacher"

For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

Input$()

Line Input

Lof()

MsgBox

Open

Read

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Read Example

This example reads each paragraph in the sequential file in turn and defines name$ as the first item and age as the second item. For each paragraph, the values are displayed in a message box.

```
Open "C:\TMP\DATA.TXT" For Input As #1
While Not Eof(1)
    Read #1, name$, age
    MsgBox name$ + " is" + Str$(age) + " years old."
Wend
Close #1
```

Read

¬Example

Read #*FileNumber***,** *Variable1*[**$**] [**,** *Variable2*[**$**]] [**,** *Variable3*[**$**]] [**,** ...]

Retrieves string or numeric values from an open sequential file, typically containing data added with the **Write** statement, and assigns the values to variables. *FileNumber* is the number specified in the **Open** instruction that opened the file for output. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

Read is similar to **Input**, but removes quotation marks from strings. For example, a text-only paragraph created with a **Write** instruction might appear as follows:

"Michelle Levine", 26,"Dancer"

Whereas **Input** interprets the first value as "Michelle Levine", **Read** takes only the text within the quotation marks: Michelle Levine.

_____

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

Input$()

Line Input

Lof()

Open

Print

Seek

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Seek Example

This example finds the first line beginning with the letter "P" in DATA.TXT and then displays a message box showing the entry and its character position in the file:

```
Open "C:\TMP\DATA.TXT" For Input As #1
Input #1, name$
While Left$(name$, 1) <> "P"
    If Eof(1) Then
        MsgBox "No P names in the file."
        Goto finish
    End If
    n = Seek(#1)
    Input #1, name$
Wend
MsgBox "First P name is " + name$ + " at position" + Str$(n)
finish:
Close #1
```

Seek, Seek()

¬Example

Seek [#]*FileNumber*, *Count*

Seek([#]*FileNumber***)**

The **Seek** statement controls where data is retrieved from a sequential file open for input, or where data is inserted into a file open for output or appending. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

Argument	Explanation
FileNumber	The number specified in the **Open** instruction that opened the file for input, output, or appending.
Count	The character position where data retrieval or insertion occurs. Note that line breaks and other delimiters, such as commas, are counted as characters.

The **Seek()** function returns the current character position in the sequential file. You can use the **Seek()** function to store the current character position before you close a sequential file, and the **Seek** statement to return to that position when you reopen the file.

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

Input$()

Line Input

Lof()

Open

Print

Read

Write

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Write Example

This example opens a sequential file for output (creating it if it does not already exist), prompts the user for three pieces of data, and then uses the **Write** statement to insert the data into the sequential file:

```
Open "C:\TMP\DATAFILE.TXT" For Output As #1
name$ = InputBox$("Enter name:")
age = Val(InputBox$("Enter age:"))
job$ = InputBox$("Enter occupation:")
Write #1, name$, age, job$
Close #1
```

The following is an example of a paragraph in DATAFILE.TXT inserted by the **Write** statement:

```
"Michelle Levine", 26,"Dancer"
```

Write

¬Example

Write #*FileNumber***,** *Expression1*[*$*] [**,** *Expression2*[*$*]] [**,** *...*]

Writes the specified expressions to an open sequential file. *FileNumber* is the number specified in the **Open** instruction that opened the file for output or appending. For more information about sequential files, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

Write is similar to the **Print** statement, but instead of separating the expressions with tab characters, **Write** separates them with commas; also, **Write** encloses strings in quotation marks. This allows the resulting values to be read by a **Read** instruction. For an illustration of the respective output of the **Print** and **Write** statements, see Print.

_____

See also

Basic File Input/Output Statements and Functions

Close

Eof()

Input

Input$()

Line Input

Lof()

Open

Print

Read

Seek

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

AddAddIn Example

This example uses **AddAddIn()** to load the template MYLIB.DOT as a global template and define the variable id as a numeric identifier for the template:

```
id = AddAddIn("C:\MYDOT\MYLIB.DOT", 1)
```

The following example fills an array with the names of functions in loaded WLLs that can be called from a macro just like Word commands:

```
nonaddins = CountMacros(0) + CountMacros(1)
loaded = CountMacros(0, 0, 1)
size = loaded - 1
Dim loaded$(size)
For count = 0 To size
    pos = (count + 1) + nonaddins
    loaded$(count) = MacroName$(pos, 0, 1)
Next count
```

AddAddIn, AddAddIn()

¬Example

AddAddIn *AddIn$* [**,** *Load*]

AddAddIn(*AddIn$* [**,** *Load*]**)**

The **AddAddIn** statement adds a template or Word add-in library (WLL) to the list of global templates and add-ins in the Templates And Add-ins dialog box (Templates command, File menu).

Argument	Explanation
AddIn$	The path and filename of the template or WLL
Load	Specifies whether to load the template or add-in after adding it to the list: 0 (zero) Does not load the template or add-in 1 or omitted Loads the template or add-in

The **AddAddIn()** function behaves the same as the statement and also returns a value corresponding to the position of the global template or add-in in the list, where 1 is the first template or add-in, 2 is the second, and so on. This value may be used with other add-in statements and functions.

You can use functions defined in a loaded WLL in a macro. Functions that take no arguments may be used just like WordBasic statements; you can return the names of these functions using **CountMacros()** and **MacroName$()**. Functions in the WLL that take arguments must be declared using the **Declare** statement.

For more information on loading global templates and add-ins, see Chapter 31, "Customizing and Optimizing Word," in the *Microsoft Word User's Guide*. For more information on using functions in WLLs, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit.

See also

Documents, Templates, and AddIns Statements and Functions

AddInState()

ClearAddIns

CountAddIns()

CountMacros()

DeleteAddIn

GetAddInID()

GetAddInName$()

MacroName$()

AddInState, AddInState()

AddInState *AddIn*, *Load*

AddInState *AddIn$*, *Load*

AddInState(*AddIn***)**

AddInState(*AddIn$***)**

The **AddInState** statement loads or unloads the specified global template or add-in listed in the Templates And Add-ins dialog box (Templates command, File menu). You can specify the global template or add-in as a number or as text.

Argument	Explanation
AddIn	A number corresponding to the global template or add-in's position in the list of global templates and add-ins, where 1 is the first template or add-in, 2 is the second, and so on
AddIn$	The path and filename of the global template or add-in
Load	Specifies whether to load or unload the global template or add-in: 0 (zero) Unloads the template or add-in 1 Loads the template or add-in

The **AddInState()** function returns the following values. Note that the return values are additive. For example, if the specified add-in is loaded, a WLL, and in the Startup directory, **AddInState()** returns 7 (the sum of 1, 2, and 4).

Value	Explanation
0 (zero)	If none of the following descriptions are true of the global template or add-in.
1	If the global template or add-in is loaded.
2	If it is a Word add-in library (WLL).
4	If the global template or add-in loads automatically; templates and add-ins in the directory specified by the Startup option on the File Locations tab in the Options dialog box (Tools menu) load automatically.

See also

Documents, Templates, and AddIns Statements and Functions

AddAddIn

ClearAddIns

CountAddIns()

DeleteAddIn

GetAddInID()
GetAddInName$()

ClearAddIns

ClearAddIns *RemoveFromList*

Unloads all global templates and Word add-in libraries (WLLs) that appear in the list of global templates and add-ins in the Templates And Add-ins dialog box (Templates command, File menu).

Argument	Explanation
RemoveFromList	Specifies whether to remove the global templates and add-ins from the list in addition to unloading them: 0 (zero) Global templates and add-ins remain in the list. 1 Global templates and add-ins are removed from the list (except for those in the Startup directory).

See also

Documents, Templates, and AddIns Statements and Functions

AddAddIn

AddInState()

CountAddIns()

DeleteAddIn

GetAddInId()

GetAddInName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Converter$() Example

This example inserts at the insertion point a list of file format class names available to Word. The instructions run until a$ is an empty string (""), indicating that the end of the list of available formats has been reached.

```
x = 0
a$ = Converter$(x)
While a$ <> ""
    Insert a$ + Chr$(13)
    x = x + 1
    a$ = Converter$(x)
Wend
```

Converter$()

¬Example

Converter$(*FormatNumber***)**

Returns the class name of the file format associated with *FormatNumber.*

Argument	Explanation
FormatNumber	A number corresponding to a file format listed under Save File As Type in the Save As dialog box (File menu): 0 (zero) corresponds to the first format, 1 to the second format, and so on. *FormatNumber* can also be a value returned by **ConverterLookup()**. Note that in some cases, the format number returned by **ConverterLookup()** does not match the position of the format name in the list box. For example, ConverterLookup("Excel Worksheet") returns 103.

See also

Documents, Templates, and AddIns Statements and Functions

ConverterLookup()

FileSaveAs

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ConverterLookup() Examples

This example saves the active document in rich-text format (RTF), using the class name "MSRTF" to specify the format:

```
FileSaveAs .Name = "C:\RTF\TEST.RTF", \
        .Format = ConverterLookup("MSRTF")
```

The following example also saves the active document in RTF, but specifies the format using the name in the Save File As Type box in the Save As dialog box instead of the class name:

```
FileSaveAs .Name = "C:\RTF\TEST.RTF", \
        .Format = ConverterLookup("Rich Text Format")
```

ConverterLookup()

¬Example

ConverterLookup(*FormatName$***)**

Returns a number corresponding to the file format specified by *FormatName$*. You can use this number with the .Format argument in a **FileSaveAs** instruction to save a file in a different format. If the specified format does not exist, **ConverterLookup()** returns -1.

Argument	Explanation
FormatName$	The class name for the format as returned by **Converter$()**, or the name of a file format as it appears in the Save File As Type box in the Save As dialog box (File menu).

See also

Documents, Templates, and AddIns Statements and Functions

Converter$()

FileSaveAs

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountAddIns() Example

This example unloads all global templates with the .DOT filename extension in the list of global templates and add-ins.

```
For i = 1 To CountAddIns()
    a$ = GetAddInName$(i)
    If InStr(a$, ".DOT") <> 0 Then
        AddInState a$, 0
    End If
Next i
```

CountAddIns()

¬Example

CountAddIns()

Returns the number of global templates and Word add-in libraries (WLLs) in the list of global templates and add-ins in the Templates And Add-ins dialog box (Templates command, File menu).

_____

See also

Documents, Templates, and AddIns Statements and Functions

AddAddIn

AddInState()

ClearAddIns

DeleteAddIn

GetAddInId()

GetAddInName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountFiles() Example

This example opens the most recently used file. You could use this instruction in an AutoExec macro. It uses **CountFiles()** to determine that at least one file is listed on the File menu.

```
If CountFiles() Then File1 Else MsgBox "Sorry, no file listed."
```

CountFiles()

¬Example

CountFiles()

Returns the number of filenames in the list of most recently used files at the bottom of the File menu.

See also

Documents, Templates, and AddIns Statements and Functions

FileList

FileName$()

File*Number*

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountFoundFiles() Example

This example determines whether the last **FileFind** search was successful:

```
If CountFoundFiles() = 0 Then MsgBox "No files found."
```

For another example, see FileFind Example.

CountFoundFiles()

¬Example

CountFoundFiles()

Returns the number of files found in the last search using **FileFind**. **CountFoundFiles()** returns 0 (zero) if no files were found in the last search or if a search has not been performed during the current Word session.

See also

Documents, Templates, and AddIns Statements and Functions

FileFind

FoundFileName$()

DeleteAddIn

DeleteAddIn *AddIn*

DeleteAddIn *AddIn$*

Removes a global template or Word add-in library (WLL) from the list of global templates and add-ins in the Templates And Add-ins dialog box (Templates command, File menu).

Argument	Explanation
AddIn	A number corresponding to the global template or add-in's position in the list, where 1 is the first template or add-in, 2 is the second, and so on.
AddIn$	The path and filename of the global template or add-in.

See also

Documents, Templates, and AddIns Statements and Functions

AddAddIn

AddInState()

ClearAddIns

CountAddIns()

GetAddInId()

GetAddInName$()

DisableInput

DisableInput [*Disable*]

Prevents the ESC key from interrupting a macro. **DisableInput** does not affect the use of ESC for canceling a dialog box.

Argument	Explanation
Disable	Determines whether or not to disable the ESC key: 0 (zero) Enables the ESC key to cancel the macro. 1 or omitted Prevents the ESC key from canceling the macro.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocClose Example

This example closes every open document window and prompts the user to save any unsaved changes:

```
While Window() <> 0
    DocClose 0
Wend
```

DocClose

¬Example

DocClose [*Save*]

Closes the active document window. **DocClose** is not the same as **FileClose**: Whereas **FileClose** closes the active document and all associated windows, **DocClose** closes only the active document window. When a document is open in a single window, which is usually the case, **DocClose** and **FileClose** have the same effect.

Argument	Explanation
Save	Determines whether or not to save the document: 0 (zero) or omitted — Prompts the user to save if changes have been made in the document since the last time it was saved. 1 — Word saves the document without prompting before closing it. 2 — Word closes the window but does not save the document.

See also

Documents, Templates, and AddIns Statements and Functions

ClosePane

FileClose

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocumentStatistics Example

This example uses the **DocumentStatistics** dialog record to obtain the number of words in the active document. You can use this example as a model for how to obtain the value of any **DocumentStatistics** argument.

```
FileSummaryInfo .Update                    'Update document statistics
Dim dlg As DocumentStatistics                  'Create a dialog record
GetCurValues dlg                           'Fill record with values
MsgBox "Number of words: " + dlg.Words
```

DocumentStatistics

¬Example

DocumentStatistics [**.FileName** = *text*] [**, .Directory** = *text*] [**, .Template** = *text*] [**, .Title** = *text*] [**, .Created** = *text*] [**, .LastSaved** = *text*] [**, .LastSavedBy** = *text*] [**, .Revision** = *text*] [**, .Time** = *text*] [**, .Printed** = *text*] [**, .Pages** = *text*] [**, .Words** = *text*] [**, .Characters** = *text*] [**, .Paragraphs** = *text*] [**, .Lines** = *text*] [**, .FileSize** = *text*]

Used with a dialog record, **DocumentStatistics** returns information about the active document. The arguments for the **DocumentStatistics** statement correspond to the information available in the Document Statistics dialog box (Summary Info command, File menu). Also, the arguments are read-only, which means that unlike other WordBasic statements, you cannot use **DocumentStatistics** to set a value. Instead, you use the **DocumentStatistics** dialog record in the same way you use a function: to return information.

Argument	Explanation
.FileName	The name of the document (the path is not included)
.Directory	The path in which the document is kept
.Template	The path and filename of the template attached to the document
.Title	The document's title
.Created	The date and time the document was created
.LastSaved	The date and time the document was last saved
.LastSavedBy	The author of the document
.Revision	The number of times the document has been saved
.Time	The total time spent editing the document
.Printed	The date and time the document was last printed
.Pages	The number of pages in the document
.Words	The number of words in the document
.Characters	The number of characters in the document
.Paragraphs	The number of paragraphs in the document
.Lines	The number of lines in the document
.FileSize	The size of the document in the form "36,352 Bytes"

Note

To ensure you get the most up-to-date information when using a **DocumentStatistics** dialog record to return document statistics, include the instruction FileSummaryInfo .Update before defining the dialog record.

See also

Documents, Templates, and AddIns Statements and Functions
FileSummaryInfo
SelInfo()

FileClose

FileClose [*Save*]

Closes the active document. All document windows containing the active document are closed. If the document is open in more than one window and you want to close only the active window, use **DocClose**.

Argument	Explanation
Save	Determines whether or not Word saves the document before closing it if the document 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 the document. 1 — Saves the document before closing it. 2 — Closes the document without saving it.

See also

Documents, Templates, and AddIns Statements and Functions

ClosePane

DocClose

FileCloseAll

FileExit

IsDocumentDirty()

FileCloseAll

FileCloseAll [*Save*]

Closes all open document windows.

Argument	Explanation
Save	Determines whether or not 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 each changed document before closing it. 2 — Closes all documents without saving changed documents.

See also

Documents, Templates, and AddIns Statements and Functions

DocClose

FileClose

FileExit

FileConfirmConversions, FileConfirmConversions()

FileConfirmConversions [*On*]

FileConfirmConversions()

The **FileConfirmConversions** statement controls whether Word displays a confirmation dialog box when a file in a format other than Word Document or Document Template is opened. **FileConfirmConversions** selects or clears the Confirm Conversions check box in the Open dialog box (File menu).

Argument	Explanation
On	Specifies whether Word displays the confirmation dialog box: 0 (zero) Does not display the dialog box. 1 Displays the dialog box. Omitted Toggles the Confirm Conversions check box.

The **FileConfirmConversions()** function returns the following values.

Value	Explanation
0 (zero)	If the Confirm Conversions check box is cleared
-1	If the Confirm Conversions check box is selected

See also

Documents, Templates, and AddIns Statements and Functions

MailMergeAskToConvertChevrons

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileFind Examples

This example searches two directories for files with the .RPT filename extension, defines an array to hold their names, and then uses **FoundFileName$()** to fill the array with the filenames.

```
FileFind .SearchPath = "C:\WINWORD; C:\REPORTS", .Name = "*.RPT"
size = CountFoundFiles() - 1
If size >= 0 Then
    Dim mydocs$(size)
    For count = 0 To size
        mydocs$(count) = FoundFileName$(count + 1)
    Next
End If
```

The following example creates a dialog record for the Find File dialog box, sets the search path and filename criteria, specifies that summary information be displayed for the found files, and then displays the list of found files.

```
Dim dlg As FileFind
dlg.SearchPath = "C:\MYDOCS"
dlg.Name = "*.DOC"
dlg.View = 2
x = Dialog(dlg)
```

FileFind

¬Example

FileFind [**.SearchName** = *text*] [**, .SearchPath** = *text*] [**, .Name =** *text*] [**, .SubDir =** *number*] [**, .Title** = text] [**, .Author** = *text*] [**, .Keywords** = *text*] [**, .Subject** = *text*] [**, .Options =** *number*] [**, .MatchCase =** *number*] [**, .Text =** *text*] [**, .PatternMatch =** *text*] [**, .DateSavedFrom =** *text*] [**, .DateSavedTo =** *text*] [**, .SavedBy =** *text*] [**, .DateCreatedFrom =** *text*] [**, .DateCreatedTo =** *text*] [**, .View =** *number*] [**, .SortBy =** *number*] [**, .ListBy =** *number*] [**, .SelectedFile =** *number*] [**, .Add**] [**, .Delete**]

Creates a list of files based on the search criteria specified by one or more of the arguments. The arguments for the **FileFind** statement correspond to the options in the Search dialog box (Find File command, File menu).

Argument	Explanation
.SearchName	A name given to a group of search criteria. You can use the name of an existing group for a search operation instead of specifying criteria one by one, or you can specify a group to create or remove (using .Add or .Delete).
.SearchPath	A list of paths in which to search for files (use semicolons to separate more than one path).
.Name	The filename of the document --- usually a file specification with a wildcard, such as "*.DOC," "*.*," or "*.BMP," although you can specify an explicit filename.
.SubDir	If 1, Word searches subdirectories of the directory or directories specified by .SearchPath.
.Title	Title in the Summary Info dialog box.
.Author	Author in the Summary Info dialog box.
.Keywords	Keywords used to identify the document in the Summary Info dialog box.
.Subject	Subject in the Summary Info dialog box.
.Options	Specifies how to list the found files: 0 (zero) Create a new list 1 Add matches to the existing file list 2 Search only in the existing file list
.MatchCase	Specifies whether to match the case of each letter in .Text: 0 (zero) Do not match the case (default) 1 Match the case
.Text	The text to search for in the document.
.PatternMatch	If 1, Word evaluates .Text as a

	string containing advanced search criteria such as the asterisk (*) and question mark (?) wildcard characters. For more information, see Advanced search criteria.
.DateSavedFrom	The document save date you want to search from.
.DateSavedTo	The document save date you want to search to.
.SavedBy	The name of the person who last saved the document.
.DateCreatedFrom	The document creation date you want to search from. The following are examples of valid date formats: 7/8/93 8-Jul-91 July 8, 1991
.DateCreatedTo	The document creation date you want to search to.
.View	Specifies what to display on the right side of the dialog box when you display the Find File dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) File information (a line for each file) 1 A preview window of the contents of the selected file 2 Summary information for the selected file
.SortBy	Specifies how documents are sorted when you display the Find File dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) Alphabetically by author 1 By creation date, with the most recently created file listed first 2 Alphabetically by the name of the person who last saved the document 3 By the date last saved, with the most recently saved file listed first 4 Alphabetically by filename 5 By size, with the smallest file listed first
.ListBy	Specifies how to list files on the left side of the dialog box when you display the Find File dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) By filename 1 By title

.SelectedFile	When you've used a **Dialog** or **Dialog()** instruction to display the Find File dialog box, returns a number when the user closes the dialog box that corresponds to the file that was last selected. To retrieve the filename, pass this value to the **FoundFileName$()** function (for example, FoundFileName$(dlg.SelectedFile)).
.Add	Stores the specified search criteria under the name specified by .SearchName; may be used to create or modify a group of search criteria.
.Delete	Removes the group of search criteria specified by .SearchName.

If you specify multiple words for .Title, .Subject, .Author, .Keywords, .Path, and .Text, Word interprets the entire argument as a logical expression rather than as a multiple-word phrase. A comma serves as the logical operator OR, and a space or an ampersand (&) serves as the logical operator AND. For more information, see Using special characters in a document search.

See also

Documents, Templates, and AddIns Statements and Functions

CountFoundFiles()

FoundFileName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileList Example

This example opens the most recently used file. You can include these instructions in an AutoExec macro so that every time you start Word, the most recently used file is opened automatically. If there is no file listed on the File menu, On Error Resume Next prevents Word from displaying an error message.

```
On Error Resume Next
FileList 1
```

For more information on auto macros, see Auto Macros.

FileList

¬Example

FileList *Number*

Opens one of the files in the list of recently used files at the bottom of the File menu. (Word defines "used" as either opened or saved.) *Number* corresponds to the number next to the filename listed on the File menu (from 1 through 9) you want to open. If *Number* is greater than the number of files listed, an error occurs. If the specified file is already open, Word activates the document.

You can control the number of filenames that appear on the File menu with the Recently Used File List option on the General tab in the Options dialog box (Tools menu).

_____

See also

Documents, Templates, and AddIns Statements and Functions

CountFiles()

FileName$()

File*Number*

FileOpen

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileName$() Example

When run, this example demonstrates the difference between **FileName$()** and **WindowName$()**. **FileName$()** always returns a file's path and filename. **WindowName$()** returns the window title, which may include just the filename or a partial path and the filename (if the document is not stored in the current directory).

```
MsgBox "File = " + FileName$() + Chr$(13) + \
    "Window = " + WindowName$()
```

FileName$()

¬Example

FileName$([*Number*]**)**

Returns the path and filename of the active document or of a file in the list of recently used files at the bottom of the File menu.

Argument	Explanation
Number	The number of the file in the order listed on the File menu, from 1 through 9. If the number is 0 (zero) or omitted, **FileName$()** returns the name of the active document; if there is no active document, it returns an empty string (""). If *Number* is greater than the number of files listed on the File menu, an error occurs. You can control the number of filenames that appear on the File menu with the Recently Used File List option on the General tab in the Options dialog box (Tools menu).

See also

Documents, Templates, and AddIns Statements and Functions

CountFiles()

FileList

FileNameFromWindow$()

File*Number*

Files$()

WindowName$()

FileNameFromWindow$()

FileNameFromWindow$([*WindowNumber*]**)**

Returns the path and filename of the document in the specified window.

Argument	Explanation
WindowNumber	The position of the window on the Window menu, where 1 is the first position, 2 is the second position, and so on. If *WindowNumber* is 0 (zero) or omitted, **FileNameFromWindow$()** returns the path and filename of the active document; if the active document has never been saved, it returns an empty string ("").

See also

Documents, Templates, and AddIns Statements and Functions

CountWindows()

FileName$()

WindowName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileNameInfo$() Example

This example uses **FileNameInfo$()** to change the filename extension of all the files in the current directory with the extension .DOC to .RTF and then saves the files in rich-text format (RTF):

```
file$ = Files$("*.DOC")
While file$ <> ""
    FileOpen .Name = file$
    noExtension$ = FileNameInfo$(file$, 4)
    FileSaveAs .Name = noExtension$ + ".RTF" , .format = 6
    FileClose 2
    file$ = Files$()
Wend
```

FileNameInfo$()

¬Example

FileNameInfo$(***Filename$*,** *InfoType***)**

Returns the part of *Filename$* specified by *InfoType*.

Argument	Explanation
Filename$	The path, if specified, and filename of a document. Though the path must exist, the filename does not have to be the name of an existing document. If the path does not exist, an error occurs.
InfoType	The part of *Filename$* to return: 1 The full path and filename. For example, FileNameInfo$(FileName$(), 1) might return "C:\DOCUMENT\TEST.DOC." If *Filename$* does not include a path, **FileNameInfo$()** returns a path made up of the current directory and *Filename$*, even if such a file does not yet exist. 2 The filename only, if *Filename$* is located in the current directory; otherwise, the full path and filename. 3 The filename. For example, FileNameInfo$("C:\DOCUMENT\TEST.DOC", 3) returns "TEST.DOC." 4 The filename without a filename extension. For example, FileNameInfo$("C:\DOCUMENT\TEST.DOC", 4) returns "TEST." 5 The full path. For example, FileNameInfo$("C:\DOCUMENT\TEST.DOC", 5) returns "C:\DOCUMENT\." 6 The universal naming convention (UNC) network path and filename. For example, if TEST.DOC is located on a server with the network share name \\DOCUMENT\PUBLIC, then FileNameInfo$("D:\DOCUMENT\TEST.DOC", 6) returns "\\DOCUMENT\PUBLIC\DOCUMENT\TEST.DOC." If *Filename$* is not located on a network share, the full path and filename are returned.

See also

Documents, Templates, and AddIns Statements and Functions

FileName$()

FileNameFromWindow$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileNew Examples

This example creates a new document based on the Memo1 template. The .DOT filename extension is optional --- you can specify either "MEMO1" or "MEMO1.DOT."

```
FileNew .Template = "MEMO1"
```

The following example creates a new template based on the existing Letter3 template:

```
FileNew .NewTemplate = 1, .Template = "LETTER3.DOT"
```

The following example runs the Resume wizard:

```
FileNew .Template = "RESUME.WIZ"
```

FileNew

¬Example

FileNew [**.Template** = *text*] [**, .NewTemplate** = *number*]

Creates a new document or template based on the template you specify, or runs a wizard. The arguments for the **FileNew** statement correspond to the options in the New dialog box (File menu).

Argument	Explanation
.Template	The name of the template or document on which to base the new document or template, or the name of the wizard to run.
.NewTemplate	Specifies whether to create a new document or a new template: 0 (zero) or omitted — Create a new document. 1 — Create a new template.

See also

Documents, Templates, and AddIns Statements and Functions

FileNewDefault

FileOpen

FileNewDefault

FileNewDefault

Creates a new document based on the Normal template.

_____

See also

Documents, Templates, and AddIns Statements and Functions

FileNew

File*Number*

File*Number*

Opens one of the files in the list of recently used files at the bottom of the File menu. (Word defines "used" as either opened or saved.) **File1** opens the first document in the list (the most recently used), **File2** opens the second document, and so on through **File9**. Nine is the maximum number of filenames that can be listed on the File menu. If no file is listed, or if *Number* is greater than the number of files listed, an error occurs. If the specified file is already open, Word activates the document.

You can control the number of files to be listed on the File menu with the Recently Used File List option on the General tab in the Options dialog box (Tools menu).

See also

Documents, Templates, and AddIns Statements and Functions

CountFiles()

FileList

FileName$()

Window*Number*

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileOpen Examples

This example changes the current directory, and then opens a read-only copy of MYDOC.DOC:

```
ChDir "C:\WINWORD\DOCS"
FileOpen .Name = "MYDOC.DOC", .ReadOnly = 1
```

The following example also opens MYDOC.DOC, but because the full path is specified, no **ChDir** instruction is required before the **FileOpen** instruction:

```
FileOpen .Name = "C:\WINWORD\DOCS\MYDOC.DOC", .ReadOnly = 1
```

FileOpen

¬Example

FileOpen .Name = *text* [**, .ConfirmConversions =** *number*] [**, .ReadOnly** = *number*] [**, .AddToMru =** *number*] [**, .PasswordDoc** = *text*] [**, .PasswordDot** = *text*] [**, .Revert =** number] [**, .WritePasswordDoc =** *text*] [**, .WritePasswordDot =** *text*]

Opens the specified document. If the document does not exist, or is not located in the specified directory, an error occurs. The arguments for the **FileOpen** statement correspond to the options in the Open dialog box (File menu).

Argument	Explanation
.Name	The name of the document (paths are accepted).
.ConfirmConversions	If 1, displays the Convert File dialog box if the file is not in Word format.
.ReadOnly	If 1, opens the document as read-only.
.AddToMru	If 1, adds the filename to the list of recently used files at the bottom of the File menu. (Note that MRU is an abbreviation for "most recently used.")
.PasswordDoc	The password to open the document, if required.
.PasswordDot	The password to open the template, if required.
.Revert	Controls what happens if .Name is the filename of an open document: 0 (zero) or omitted — Word activates the open document. 1 — Word discards any unsaved changes to the open document and reopens the file.
.WritePasswordDoc	The password to save changes to the document, if required.
.WritePasswordDot	The password to save changes to the template, if required.

See also

Documents, Templates, and AddIns Statements and Functions

FileConfirmConversions

FileFind

FileNew

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FilePageSetup Examples

This example sets the top margin to 1 inch for the entire document:

```
FilePageSetup .ApplyPropsTo = 4, .TopMargin = "1 in"
```

The following example displays the Paper Source tab in the Page Setup dialog box, and then applies the settings the user selects:

```
Dim dlg As FilePageSetup
GetCurValues dlg
dlg.Tab = 2
Dialog dlg
FilePageSetup dlg
```

FilePageSetup

¬Example

FilePageSetup [**.Tab** = *number*] [**, .TopMargin** = *number or text*] [**, .BottomMargin** = *number or text*] [**, .LeftMargin** = *number or text*] [**, .RightMargin** = *number or text*] [**, .Gutter** = *number or text*] [**, .PageWidth** = *number or text*] [**, .PageHeight** = *number or text*] [**, .Orientation** = *number*] [**, .FirstPage** = *number*] [**, .OtherPages** = *number*] [**, .VertAlign** = *number*] [**, .ApplyPropsTo** = *number*] [**, .Default**] [**, .FacingPages** = *number*] [**, .HeaderDistance** = *number or text*] [**, .FooterDistance** = *number or text*] [**, .SectionStart** = *number*] [**, .OddAndEvenPages** = *number*] [**, .DifferentFirstPage** = *number*] [**, .Endnotes** = *number*] [**, .LineNum** = *number*] [**, .StartingNum** = *number*] [**, .FromText** = *number or text*] [**, .CountBy** = *number*] [**, .NumMode** = *number*]

Sets page attributes such as margins and page width for the entire document or sections within the document. The arguments for the **FilePageSetup** statement correspond to the options in the Page Setup dialog box (File menu).

Argument	Explanation
.Tab	Specifies which tab to select when you display the Page Setup dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) Margins 1 Paper Size 2 Paper Source 3 Layout For details, see the second example in this entry.
.TopMargin	The distance between the top edge of the page and the top boundary of the body text in points or a text measurement.
.BottomMargin	The distance between the bottom edge of the page and the bottom boundary of the body text in points or a text measurement.
.LeftMargin	The distance between the left edge of the page and the left boundary of the body text in points or a text measurement.

.RightMargin	The distance between the right edge of the page and the right boundary of the body text in points or a text measurement.
.Gutter	The extra margin space allowed for binding the document, in points or a text measurement.
.PageWidth	The width of the page in points or a text measurement.
.PageHeight	The height of the page in points or a text measurement.
.Orientation	The orientation of the page: 0 (zero) Portrait 1 Landscape Note that unlike using the TogglePortrait statement, specifying a new orientation with FilePageSetup does not automatically adjust the page size and margins for the new orientation.
.FirstPage, .OtherPages	Selects the paper source for the first page and the other pages in the document: 0 (zero) Default Tray (determined by the printer driver) 1 Upper Tray 2 Lower Tray 4 Manual Feed (often used to override the default tray for the first page) 5 Envelope Other values may be available

	depending on your printer driver. To determine a value, record a macro that selects the option you want on the Paper Source tab in the Page Setup dialog box, and then review the values for .FirstPage and .OtherPages in the macro-editing window.
.VertAlign	Alignment of section on the page: 0 (zero) Top 1 Center 2 Justified
.ApplyPropsTo	The part of the document to apply the page setup properties to: 0 (zero) This Section 1 This Point Forward 2 Selected Sections 3 Selected Text 4 Whole Document
.Default	Makes the current page-setup properties the default for new documents based on the active template.
.FacingPages	If 1, corresponds to selecting the Mirror Margins check box.
.HeaderDistance	Distance of header from the top of the page.
.FooterDistance	Distance of footer from the bottom of the page.
.SectionStart	Determines the type of section break: 0 (zero) Continuous 1 New Column

	2 New Page 3 Even Page 4 Odd Page
.OddAndEvenPages	If 1, corresponds to selecting the Different Odd And Even check box.
.DifferentFirstPage	If 1, corresponds to selecting the Different First Page check box.
.Endnotes	If 1, corresponds to selecting the Suppress Endnotes check box.
.LineNum	If 1, corresponds to selecting the Add Line Numbering check box.
.StartingNum	The number at which to begin line numbering.
.FromText	The distance from text, in points or a text measurement; 0 (zero) sets automatic spacing.
.CountBy	The numeric increment used to print line numbers.
.NumMode	Determines how lines are numbered: 0 (zero) Restart at each new page 1 Restart at each new section 2 Continuous

See also

Documents, Templates, and AddIns Statements and Functions

FormatColumns

FormatSectionLayout

FilePrint

FilePrint [**.Background** = *number*] [**, .AppendPrFile** = *number*] [**, .Range** = *number*] [**, .PrToFileName** = *text*] [**, .From** = *text*] [**, .To** = *text*] [**, .Type** = *number*] [**, .NumCopies** = *number*] [**, .Pages** = *text*] [**, .Order** = *number*] [**, .PrintToFile** = *number*] [**, .Collate** = *number*] [**, .FileName** = *text*]

Prints all or part of the active document or a document you specify. The arguments for the **FilePrint** statement correspond to the options in the Print dialog box (File menu).

Argument	Explanation
.Background	If 1, the macro continues while Word prints the document.
.AppendPrFile	If you print to a file, specifies whether to overwrite or append to the file if it already exists: 0 (zero) Overwrite 1 Append
.Range	The page range: 0 (zero) Prints the entire document 1 Prints the selection 2 Prints the current page 3 Prints the range of pages specified by .From and .To 4 Prints the range of pages specified by .Pages
.PrToFileName	If you print to a file, the path and filename of the file to print to.
.From	The starting page number when .Range is 3.
.To	The ending page number when .Range is 3.
.Type	The item to print: 0 (zero) Document 1 Summary Info 2 Annotations 3 Styles 4 AutoText Entries 5 Key Assignments
.NumCopies	The number of copies to print.
.Pages	The page numbers and page ranges you want to print, separated by commas. For example, "2, 4-10" prints page 2 and pages 4 through 10.
.Order	Further delimits the range of pages to print: 0 (zero) Prints all pages in the range. 1 Prints only odd pages in the range. 2 Prints only even pages in the range.
.PrintToFile	If 1, sends printer instructions to a file. Make sure to specify a filename with .PrToFileName.

.Collate	If 1, organizes pages when printing multiple copies of the document.
.FileName	The path and filename of the document to print. If omitted, Word prints the active document.

See also

Documents, Templates, and AddIns Statements and Functions

FilePrintDefault

FilePrintSetup

ToolsOptionsPrint

FilePrintDefault

FilePrintDefault

Prints the active document using the current settings in the Print and Print Setup dialog boxes (File menu) and on the Print tab in the Options dialog box (Tools menu).

_____

See also

Documents, Templates, and AddIns Statements and Functions

FilePrint

FilePrintSetup

ToolsOptionsPrint

FilePrintPreview, FilePrintPreview()

FilePrintPreview [*On*]

FilePrintPreview()

The **FilePrintPreview** statement switches the active document to and from print preview.

Argument	Explanation
On	Displays the document in print preview or the previous view: Omitted Toggles print preview. 0 (zero) Cancels print preview. 1 Switches to print preview.

The **FilePrintPreview()** function returns the following values.

Value	Explanation
-1	If the active document is in print preview
0 (zero)	If the active document is in some other view

See also

Documents, Templates, and AddIns Statements and Functions

ClosePreview

FilePrintPreviewFullScreen

FilePrintPreviewPages

ViewZoom

FilePrintPreviewFullScreen

FilePrintPreviewFullScreen

Toggles the display of rulers, scroll bars, the status bar, and the menu bar in print preview. If the active document is not in print preview, an error occurs.

See also

Documents, Templates, and AddIns Statements and Functions

ToolsOptionsView

FilePrintPreviewPages, FilePrintPreviewPages()

FilePrintPreviewPages *PagesAcross*

FilePrintPreviewPages()

The **FilePrintPreviewPages** statement specifies whether to display one or two columns of pages in page layout view and print preview. *PagesAcross* can be 1 or 2 only; for more control over the grid of page images, use **ViewZoom**. The **FilePrintPreviewPages()** function returns the number of columns of pages currently displayed. If the active document is not in either page layout view or print preview, an error occurs.

See also

Documents, Templates, and AddIns Statements and Functions

FilePrintPreview

ViewPage

ViewZoom

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FilePrintSetup Examples

This example changes to the PostScript printer attached to the COM2 port:

```
FilePrintSetup .Printer = "PostScript printer on COM2:"
```

The following example uses the **SendKeys** statement to send the key sequence ALT+G, 75, ENTER to the Options dialog box for the current printer. The instructions depend on the existence of a Scaling box whose access key is G to set the scaling to 75 percent.

```
SendKeys "%g75{enter}"
FilePrintSetup .Options
```

FilePrintSetup

¬Example

FilePrintSetup [**.Printer** = *text*] [**, .Options**]

Changes the active printer or its options. The arguments for the **FilePrintSetup** statement correspond to the options in the Print Setup dialog box (Print command, File menu).

Argument	Explanation
.Printer	The name of the new printer to be activated. Specify this argument exactly as it appears in the Printer Setup dialog box.
.Options	Displays a dialog box showing the options for the current printer. You can use the **SendKeys** statement to select options.

See also

Documents, Templates, and AddIns Statements and Functions

FilePrint

SendKeys

ToolsOptionsPrint

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileRoutingSlip Example

This example adds a routing slip to the active document (which includes the current date in the subject line), adds two recipients, and then routes the document:

```
curdate$ = Date$(Now())
FileRoutingSlip .Subject = "Status Doc " + curdate$, \
    .Message = "Please fill in your status.", .AllAtOnce = 0, \
    .ReturnWhenDone = 1, .TrackStatus = 1, .Protect = 1, .AddSlip
FileRoutingSlip .Address = "Sara Levine", .AddRecipient
FileRoutingSlip .Address = "Carlos Alicea", .AddRecipient
FileRoutingSlip .RouteDocument
```

FileRoutingSlip

¬Example

FileRoutingSlip [**.Subject =** *text*] [**, .Message =** *text*] [**, .AllAtOnce =** *number*] [**, .ReturnWhenDone =** *number*] [**, .TrackStatus =** *number*] [**, .Protect =** *number*] [**, .AddSlip**] [**, .RouteDocument**] [**, .AddRecipient**] [**, .OldRecipient**] [**, .ResetSlip**] [**, .ClearSlip**] [**, .ClearRecipients**] [**, .Address =** *text*]

Adds, removes, or changes the routing slip for the active document or routes the active document to the recipients specified by the routing slip. The arguments for **FileRoutingSlip** correspond to the options in the Routing Slip dialog box (Add Routing Slip command, File menu).

Argument	Explanation
.Subject	Text for the subject line of the electronic mail message.
.Message	The message you want to precede the icon for the attached document.
.AllAtOnce	Specifies how recipients receive the document: 0 (zero) Sends the document to the first recipient specified by .Address. 1 Sends a copy of the document to all the recipients at the same time.
.ReturnWhenDone	If 1, sends the document back to the original sender when the last recipient chooses the Send command from the File menu.
.TrackStatus	If 1, sends a message back to the original sender each time the document is forwarded.
.Protect	Specifies a level of protection for the document: 0 (zero) No protection. 1 All changes are tracked by revision marks. 2 Recipients can add annotations only. 3 Recipients can enter information in form fields only.
.Address	The address of a recipient for the document you are routing. To add multiple recipients, use multiple **FileRoutingSlip** instructions to add them one at a time. If you are routing the document to recipients one at a time, add recipient names in the order you want to route the document.

Only one of the following arguments can be specified for each **FileRoutingSlip** instruction.

Argument	Explanation
.AddSlip	Adds a routing slip to the active document. Note that when you close a

	document with an attached routing slip, Word displays a message asking if you want to route the document. To suppress this message in a macro that contains **DocClose**, **FileClose**, **FileCloseAll**, or **FileExit** instructions, set the .Save argument for those statements to 1 or 2. Word closes the document or documents without routing them.
.RouteDocument	Routes the active document.
.AddRecipient	Adds the address specified by .Address to the list of recipients.
.OldRecipient	Adds the address specified by .Address to the list of recipients only if the document has not already been routed. When you record a modification to a routing slip for a document that has already been routed, **FileRoutingSlip** instructions with .OldRecipient arguments are recorded for recipients who have already received the document.
.ResetSlip	Prepares the document to be rerouted to the recipients. If the document has not completed its first routing, Word displays a message.
.ClearSlip	Removes the routing slip from the active document.
.ClearRecipients	Removes all addresses from the list of recipients.

See also

Documents, Templates, and AddIns Statements and Functions

FileSendMail

FileSave

FileSave

Saves the active document or template. If the document is not named, **FileSave** displays the Save As dialog box (File menu).

_____

See also

Documents, Templates, and AddIns Statements and Functions

FileSaveAll

FileSaveAs

IsDocumentDirty()

IsTemplateDirty()

FileSaveAll

FileSaveAll [*Save*] [**,** *OriginalFormat*]

Saves all changed files, including the Normal template and any other templates attached to open documents that have changed. It's a good idea to include a **FileSaveAll** statement if your macro makes changes to AutoText entries, other macros, or keyboard, menu, and toolbar assignments.

Argument	Explanation
Save	Specifies whether or not to prompt the user to save each document or template if it is "dirty" --- that is, if changes have been made since the last time the document or template was saved: 0 (zero) Prompts the user to save each changed document or template. 1 Saves all changed documents and templates automatically.
OriginalFormat	Specifies whether or not to prompt when saving a document opened from a foreign file format in which changes have been made that cannot be stored in the original format: 0 (zero) Saves the document in Word Document format. 1 Saves the document in its original format. 2 or omitted Prompts the user to save in Word Document format.

See also

Documents, Templates, and AddIns Statements and Functions

FileSave

FileSaveAs

IsDocumentDirty()

IsTemplateDirty()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileSaveAs Example

This example saves the active document in rich-text format (RTF) with the filename TEST.RTF:

```
FileSaveAs .Name = "TEST.RTF", .Format = 6
```

FileSaveAs

¬Example

FileSaveAs [**.Name** = *text*] [**, .Format** = *number*] [**, .LockAnnot** = *number*] [**, .Password** = *text*] [**, .AddToMru =** *number*] [**, .WritePassword =** *text*] [**, .RecommendReadOnly =** *number*] [**, .EmbedFonts =** *number*] [**, .NativePictureFormat =** *number*] [**, .FormsData =** *number*]

Saves the active document with a new name or format. The arguments for the **FileSaveAs** statement correspond to the options in the Save As dialog box (File menu).

Argument	Explanation
.Name	The new name. If you don't specify .Name, the current directory and name are the defaults. If the document has never been saved, Word saves to a default name such as DOC1.DOC for Document1, DOC2.DOC for Document2, and so on. Note that if a document with a default name already exists, Word overwrites that document without prompting.
.Format	Specifies the new format: 0 (zero) Word document 1 Document template 2 Text Only (extended characters saved in ANSI character set) 3 Text Only with Line Breaks (extended characters saved in ANSI character set) 4 MS-DOS Text (extended characters saved in IBM PC character set) 5 MS-DOS Text with Line Breaks (extended characters saved in IBM PC character set) 6 Rich-Text Format (RTF) You can specify other file formats, which must be listed in the [MSWord Text Converters] section of the WIN.INI file. To determine the appropriate value, count down, starting at 0 (zero), to the file format you want, and then add 100.
.LockAnnot	If 1, locks the document for annotations. You can also lock a document with **ToolsProtectDocument**.
.Password	Sets a password for opening the document.
.AddToMru	If 1, places the document name first on the list of recently used files at the bottom of the File menu. (Note that MRU is an abbreviation for "most recently

	used.")
.WritePassword	Sets a password for saving changes to the document.
.RecommendReadOnly	If 1, displays a message upon opening the document suggesting that it be opened as read-only.
.EmbedFonts	If 1, embeds TrueType fonts in the document.
.NativePictureFormat	If 1, saves only the Windows version of imported graphics.
.FormsData	If 1, saves the data entered by a user in a form as a data record. Note the form must be unprotected or **FileSaveAs** generates an error.

See also

Documents, Templates, and AddIns Statements and Functions

Converter$()

ConverterLookup()

FileSave

FileSaveAll

Name

ToolsOptionsSave

ToolsProtectDocument

FileSendMail

FileSendMail

Opens a message window for sending the active document through Microsoft Mail. Use the **ToolsOptionsGeneral** statement with the .SendMailAttach argument to control whether the document is sent as text in the message window or as an attachment.

_____

See also

Documents, Templates, and AddIns Statements and Functions

ToolsOptionsGeneral

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FileSummaryInfo Examples

This example sets the title of the active document:

```
FileSummaryInfo .Title = "Exploration of the Upper Amazon."
```

The following example defines the variable subject$ with the text in the Subject box in the Summary Info dialog box (File menu).

```
Dim dlg As FileSummaryInfo
GetCurValues dlg
subject$ = dlg.Subject
```

FileSummaryInfo

¬Example

FileSummaryInfo [**.Title** = *text*] [**, .Subject** = *text*] [**, .Author** = *text*] [**, .Keywords** = *text*] [**, .Comments** = *text*] [**, .FileName** = *text*] [**, .Directory** = *text*] [**, .Template** = *text*] [**, .CreateDate** = *text*] [**, .LastSavedDate** = *text*] [**, .LastSavedBy** = *text*] [**, .RevisionNumber** = *text*] [**, .EditTime** = *text*] [**, .LastPrintedDate** = *text*] [**, .NumPages** = *text*] [**, .NumWords** = *text*] [**, .NumChars** = *text*] [**, .NumParas** = *text*] [**, .NumLines** = *text*] [**, .Update**] [**, .FileSize** = *text*]

Sets the summary information for the active document and allows access to the information in the Document Statistics dialog box. The arguments for the **FileSummaryInfo** statement correspond to the options in the Summary Info dialog box (File menu) and the Document Statistics dialog box (Statistics button, Summary Info command). All the arguments corresponding to the options in the Document Statistics dialog box are read-only, with the exception of .EditTime.

Argument	Explanation
.Title	The title of the document
.Subject	The subject of the document
.Author	The author of the document
.Keywords	The keywords used to identify the document
.Comments	The comments about the document
.FileName	When using **FileSummaryInfo** as a statement, the name of the file to make summary information changes to (the file must be open). When returning information from a **FileSummaryInfo** dialog record, the filename of the active document (path not included)
.Directory	The document's directory location (read-only)
.Template	The document template (read-only)
.CreateDate	The creation date (read-only)
.LastSavedDate	The date the document was last saved (read-only)
.LastSavedBy	The name of the person last saving the document (read-only)
.RevisionNumber	The number of times the document has been saved (read-only)
.EditTime	The total time the document has been open, in minutes
.LastPrintedDate	The date the document was last printed (read-only)
.NumPages	The number of pages (read-only)
.NumWords	The number of words (read-only)
.NumChars	The number of characters (read-only)

.NumParas	The number of paragraphs (read-only)
.NumLines	The number of lines (read-only)
.Update	Updates the summary information
.FileSize	The size of the document in the form "36,352 Bytes"

Note

To ensure you get the most up-to-date information when using a **FileSummaryInfo** dialog record to return summary information or document statistics, include the instruction FileSummaryInfo .Update before defining the dialog record.

See also

Documents, Templates, and AddIns Statements and Functions

DocumentStatistics

FileTemplates

FileTemplates [**.Template** = *text*] [**, .LinkStyles =** *number*]

Changes the template attached to the active document. To add or remove global templates, use **AddAddIn**, **AddInState**, and **DeleteAddIn**. The arguments for the **FileTemplate** statement correspond to the options in the Templates And Add-ins dialog box (Templates command, File menu).

Argument	Explanation
.Template	The path and filename of the template to attach.
.LinkStyles	If 1, styles in the active document are linked to the active template --- Word copies styles from the attached template each time the document is opened.

See also

Documents, Templates, and AddIns Statements and Functions

AddAddIn

AddInState

DeleteAddIn

FileNew

Organizer

FoundFileName$()

FoundFileName$(*Number***)**

Returns the name of a file found in the most recent **FileFind** operation. The argument *Number* is a number between 1 and **CountFoundFiles()**. For an example, see FileFind Example.

See also

Documents, Templates, and AddIns Statements and Functions

CountFoundFiles()

FileFind

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetAddInID() Example

This example displays one of two messages, depending on whether the specified template is currently loaded:

```
id = GetAddInID("C:\TEMPLATES\MYDOT.DOT")
If id > 0 Then
    MsgBox "MYDOT.DOT is global template number" + Str$(id) + "."
Else
    MsgBox "MYDOT.DOT is not currently a global template."
End If
```

GetAddInID()

¬Example

GetAddInID(*AddIn$***)**

Returns a number corresponding to the specified global template or add-in's position in the list of global templates and add-ins in the Templates And Add-ins dialog box (Templates command, File menu): 1 corresponds to the first template or add-in, 2 to the second, and so on. If *AddIn$* is not currently a global template or add-in, **GetAddInID()** returns 0 (zero).

Argument	Explanation
AddIn$	The path and filename of the global template or Word add-in library (WLL).

See also

Documents, Templates, and AddIns Statements and Functions

AddAddIn

AddInState

ClearAddIns

CountAddIns()

DeleteAddIn

GetAddInName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetAddInName$() Example

This example displays a list of the current global templates and add-ins in a message box:

```
size = CountAddIns() - 1
Dim GlobalAddIns$(size)
For i = 0 To size
    GlobalAddIns$(i) = GetAddInName$(i + 1)
    msg$ = msg$ + GlobalAddIns$(i) + Chr$(13)
Next
MsgBox msg$, "Current Global Templates and Add-Ins"
```

GetAddInName$()

¬Example

GetAddInName$(*AddInID***)**

Returns the path and filename of the global template or add-in whose position in the list of global templates and add-ins in the Templates And Add-ins dialog box (Templates command, File menu) corresponds to *AddInID*: 1 corresponds to the first template or add-in, 2 to the second, and so on.

See also

Documents, Templates, and AddIns Statements and Functions

AddAddIn

AddInState

ClearAddIns

CountAddIns()

DeleteAddIn

GetAddInID()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetDocumentVar$() Example

This AutoOpen macro determines whether there is a reminder note and, if so, displays it:

```
Sub MAIN
worknote$ = GetDocumentVar$("reminder")
If worknote$ <> "" Then
    MsgBox worknote$, "Note from last time"
End If
End Sub
```

For an example that prompts for a reminder note to store with a document before closing it, see SetDocumentVar Example.

GetDocumentVar$()

¬Example

GetDocumentVar$(***VariableName$*)**

Returns the string associated with *VariableName$*, which was previously set for the active document with the **SetDocumentVar** statement. If the insertion point is not in a document --- for example, if the macro-editing window is active --- an error occurs.

See also

Documents, Templates, and AddIns Statements and Functions

SetDocumentVar

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertFile Examples

This example inserts the contents of PRICES.DOC into the active document:

```
InsertFile .Name = "C:\PRICES\PRICES.DOC"
```

The following example inserts the portion of PRICES.DOC marked with the bookmark "sportscars" into the active document:

```
InsertFile .Name = "PRICES.DOC", .Range = "sportscars"
```

The following example creates a new document and then inserts one INCLUDETEXT field for each file in the directory C:\TMP with the filename extension .TXT:

```
FileNewDefault
ChDir "C:\TMP"
name$ = Files$("*.TXT")
While name$ <> ""
    InsertFile name$, .Link = 1, .ConfirmConversions = 0
    InsertPara
    name$ = Files$()
Wend
```

InsertFile

¬Example

InsertFile .Name = *text* [**, .Range =** *text*] [**, .ConfirmConversions =** *number*] [**, .Link =** *number*]

Inserts all or part of the specified file at the insertion point.

Argument	Explanation
.Name	The path and filename of the file to insert. If the path is not specified, the current directory is assumed.
.Range	If the file specified by .Name is a Word document, .Range is a bookmark. If the file is another type (for example, a Microsoft Excel worksheet), .Range refers to a named range or cell range (for example, R1C1:R3C4). If Word cannot recognize .Range, an error occurs.
.ConfirmConversions	Specifies whether Word displays the Convert File dialog box when inserting files in formats other than Word Document format.
.Link	If 1, inserts an INCLUDETEXT field instead of the contents of the file itself.

See also

Documents, Templates, and AddIns Statements and Functions

InsertDatabase

InsertField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Name Examples

This example uses **ChDir** to set the current directory, and then renames the file COGS.DOC as COGS88.DOC in that directory. Without the **ChDir** instruction preceding it, the **Name** instruction would need to specify the complete path for each filename.

```
ChDir "C:\MYDOCS\MEMOS\JULY"
Name "COGS.DOC" As "COGS88.DOC"
```

The following example moves the file COGS.DOC to a different directory and leaves the filename unchanged:

```
Name "C:\TMP\COGS.DOC" As "C:\DOCS\COGS.DOC"
```

Name

¬Example

Name *OldName$* **As** *NewName$*

Renames or moves a file. If you do not include paths with the *OldName$* and *NewName$* arguments, **Name** assumes the current directory --- the directory selected in the Open dialog box (File menu). By including a different path for *NewName$*, you can move a file to a different directory. You cannot use **Name** to move directories or to move a file to a different drive.

Argument	Explanation
OldName$	The previous name of the file. If the filename specified by *OldName$* does not exist or is open, an error occurs.
NewName$	The new name of the file. If the filename specified by *NewName$* already exists, an error occurs.

The *OldName$* and *NewName$* arguments do not accept wildcard characters --- the asterisk (*) and question mark (?).

See also

Documents, Templates, and AddIns Statements and Functions

ChDir

CopyFile

FileSaveAs

Kill

MkDir

RmDir

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Organizer Example

This macro copies the AutoText entries from the active template to the Normal template. First, the name of the active template is retrieved from the Templates And Add-ins dialog box (Templates command, File menu). If the Document Template text box is blank, indicating the active document is a template, the template's filename is retrieved from the Document Statistics dialog box (Statistics button, Summary Info command, File menu). Then, the number of AutoText entries stored in the active template is returned. Finally, using a **For...Next** loop, the **Organizer** instruction is repeated once for each AutoText entry.

```
Sub MAIN
Dim dlg As FileTemplates
GetCurValues dlg
template$ = dlg.Template
If template$ = "" Then
    Redim dlg As DocumentStatistics
    GetCurValues dlg
    template$ = dlg.Directory + "\" + dlg.FileName
End If
MsgBox template$
num = CountAutoTextEntries(1)
If num = 0 Then Goto bye
For count = 1 To num
    Organizer .Copy, .Source = template$, \
        .Destination = "C:\WINWORD\TEMPLATE\NORMAL.DOT", \
        .Name = AutoTextName$(count, 1), .Tab = 1
Next count
bye:
End Sub
```

Organizer

¬Example

Organizer [**.Copy,**] [**.Delete,**] [**.Rename,**] [**.Source =** *text***,**] [**.Destination =** *text***,**] **.Name =** *text* [**, .NewName =** *text*] [**, .Tab =** *number*]

Deletes and renames styles, AutoText entries, toolbars, and macros, and copies these elements between templates. The arguments for the **Organizer** statement correspond to the options in the Organizer dialog box (Macro command, Tools menu).

Argument	Explanation
.Copy	Copies the specified item from the source to the destination.
.Delete	Deletes the specified item from the source.
.Rename	Renames the specified item in the source.
.Source	The filename of the document or template containing the item you want to copy, delete, or rename. Paths are accepted.
.Destination	The filename of the document or template to which you want to copy the item. Paths are accepted.
.Name	The name of the style, AutoText entry, toolbar, or macro you want to copy, delete, or rename.
.NewName	A new name for the specified item; used with .Rename.
.Tab	The kind of item you want to copy, delete, or rename: 0 (zero) Styles 1 AutoText 2 Toolbars 3 Macros You also use these values to specify which tab to select when you display the Organizer dialog box with a **Dialog** or **Dialog()** instruction.

Note

When you use **Organizer** to make a change to a file that is not open, Word opens the file in the background, saves changes, and then closes the file. If you need to make a large number of changes, your macro will run faster if it opens the file first, performs the **Organizer** instructions, and then closes the file using the instruction FileClose 1 to save changes.

See also

Documents, Templates, and AddIns Statements and Functions

EditAutoText

FileTemplates

FormatStyle

NewToolbar
ToolsMacro

SaveTemplate

SaveTemplate

Saves changes to the active template unless the active template is the Normal template. If your macro includes changes to items such as AutoText entries or keyboard, toolbar, and menu assignments for a template other than Normal, it's a good idea to include a **SaveTemplate** instruction to preserve the changes. To save changes to all documents and templates, including the Normal template, use **FileSaveAll**.

SaveTemplate does not request confirmation before saving changes. You may want to use **MsgBox()** to confirm that changes should be saved before running **SaveTemplate**.

_____

See also

Documents, Templates, and AddIns Statements and Functions

FileSave

FileSaveAll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetDocumentVar Example

This example prompts for a reminder note to store with a document before closing it. If the macro were named FileClose, it would run each time the user chose Close from the File menu (the instruction FileClose runs the built-in command).

The instruction On Error Goto CloseNow ensures that a WordBasic error doesn't appear if the user cancels the **InputBox$()** dialog box. The instructions Err = 0 and On Error Resume Next reset error handling and ensure that an error doesn't appear if the user cancels the prompt to save changes.

```
On Error Goto CloseNow
worknote$ = InputBox$("Type a note for next time:")
SetDocumentVar "reminder", worknote$
CloseNow:
Err = 0
On Error Resume Next
FileClose
```

For an example of an AutoOpen macro that displays the most recent reminder note each time the document is opened, see GetDocumentVar$() Example.

SetDocumentVar, SetDocumentVar()

¬Example

SetDocumentVar *VariableName$*, *VariableText$*

SetDocumentVar(*VariableName$*, *VariableText$***)**

The **SetDocumentVar** statement associates the string *VariableText$* with the active document. You use the *VariableName$* argument with the function **GetDocumentVar$()** to return the associated string when the document is active. The variable is saved with the document. You can set multiple document variables for a single document. If the insertion point is not in a document --- for example, if the macro-editing window is active --- an error occurs.

The **SetDocumentVar()** function behaves the same as the statement, and also returns -1 if the variable is set successfully.

See also

Documents, Templates, and AddIns Statements and Functions

GetDocumentVar$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsFileLocations Example

This example sets the default directory for documents to C:\MYDOCS:

```
ToolsOptionsFileLocations .Path = "DOC-PATH", \
        .Setting = "C:\MYDOCS"
```

ToolsOptionsFileLocations

¬Example

ToolsOptionsFileLocations .Path = *text* **, .Setting =** *text*

Sets default directories. The arguments for the **ToolsOptionsFileLocations** statement correspond to the options on the File Locations tab in the Options dialog box (Tools menu). The new setting takes effect immediately.

Argument	Explanation
.Path	One of the following lines in the [Microsoft Word] section of your WINWORD6.INI file (use this text): DOC-PATH PICTURE-PATH USER-DOT-PATH WORKGROUP-DOT-PATH INI-PATH AUTOSAVE-PATH TOOLS-PATH CBT-PATH STARTUP-PATH
.Setting	The path for the default directory, or an empty string ("") to remove the line from WINWORD6.INI.

See also

Documents, Templates, and AddIns Statements and Functions

SetPrivateProfileString

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditFind Examples

This example finds the next instance of "Trey Research" with single-underline formatting:

```
EditFindClearFormatting                    'Clear leftover formats
EditFindFont .Underline = 1
EditFind .Find = "Trey Research", .Direction = 0, \
    .WholeWord = 0, .MatchCase = 0, .Format = 1
```

The following example inserts the string "Tip: " at the beginning of every Heading 6 paragraph. The instructions StartOfDocument and While EditFindFound() are key to any macro that repeats a series of actions each time the specified item is found.

```
StartOfDocument
EditFindClearFormatting
EditFindStyle .Style = "Heading 6"
EditFind .Find = "", .Direction = 0, .Format = 1, .Wrap = 0
While EditFindFound()
    StartOfLine
    Insert "Tip: "
    EditFindStyle .Style = "Heading 6"
    EditFind .Find = "", .Direction = 0, .Format = 1, .Wrap = 0
Wend
```

EditFind

¬Example

EditFind [**.Find** = *text*] [**, .Direction** = *number*] [**, .WholeWord** = *number*] [**, .MatchCase** = *number*] [**, .PatternMatch** = *number*] [**, .SoundsLike** = *number*] [**, .Format** = *number*] [**, .Wrap** = *number*]

Finds the next instance of the specified text, formatting, or both. The arguments for the **EditFind** statement correspond to the options in the Find dialog box (Edit menu). Used in a **While...Wend** loop, **EditFind** can be extremely useful when you need to repeat a series of instructions each time a certain piece of text or formatting is found in your document. Many examples in WordBasic Help illustrate this common use of **EditFind**.

Argument	Explanation
.Find	The text to find, or, to search for formatting only, an empty string (""). You can search for special characters, such as paragraph marks, by specifying appropriate character codes. For example, "^p" corresponds to a paragraph mark and "^t" corresponds to a tab character. For a complete list of special characters, see Finding and replacing special characters by using keyboard codes. If .PatternMatch is set to 1, you can specify wildcard characters and other advanced search criteria. For example, "*(ing)" finds any word ending in "ing." For more information, see Advanced Search Criteria To search for a symbol character, specify the symbol's ANSI value following a caret (^) and a zero (0). For example, "^0151" corresponds to an em dash (---). For a table of symbols and their ANSI values, see Appendix A, "Character Sets," in the *Microsoft Word User's Guide*.
.Direction	The direction to search: 0 (zero) Searches toward the end of the document. 1 Searches toward the beginning of the document. The default is the direction used in the previous search or 0 (zero) the first time the Find or Replace command is run.
.WholeWord	If 1, corresponds to selecting the Find Whole Words Only check box.
.MatchCase	If 1, corresponds to selecting the Match Case check box.

.PatternMatch	If 1, Word evaluates .Find as a string containing advanced search criteria such as the asterisk (*) and question mark (?) wildcard characters (corresponds to selecting the Use Pattern Matching check box).
.SoundsLike	If 1, corresponds to selecting the Sounds Like check box.
.Format	Finds formatting in addition to, or instead of, text: 0 (zero) Ignores formatting. 1 Uses the formatting specified by **EditFindFont**, **EditFindLang**, **EditFindPara**, or **EditFindStyle**.
.Wrap	Controls what happens if the search begins at a point other than the beginning of the document and the end of the document is reached (or vice versa if .Direction is set to 1). The .Wrap argument also controls what happens if there is a selection and the search text is not found in the selection. 0 (zero) or omitted The search operation ends and the macro continues. 1 If there is a selection, Word searches the remainder of the document. If the beginning or the end of the document is reached, Word continues the search from the opposite end. 2 If there is a selection, Word displays a message asking whether to search the remainder of the document. If the beginning or the end of the document is reached, Word displays a message asking whether to continue the search from the opposite end.

Formatting is not specified within the **EditFind** instruction itself. Instead, you precede **EditFind** with one or more **EditFindFont**, **EditFindLang**, **EditFindPara**, and **EditFindStyle** instructions. Then, in the **EditFind** instruction, you set .Format to 1.

Here are some tips for using **EditFind** effectively:

¬ **EditFind** retains whatever settings were used in the most recent find or replace operation. For example, if the Find Whole Words Only check box was selected in the Find dialog box in the last find operation (or if the .WholeWord argument was set to 1), .WholeWord will still be set to 1 in the new find operation---unless you explicitly set it to 0 (zero). Before beginning a find operation, make sure to specify every option that can affect the outcome of a search.

¬ When you search for formatting, it's a good idea to include the statement **EditFindClearFormatting** before specifying the formats you want to find. That way, you clear any formats left over from previous find or replace operations.

¬ When repeating an operation for each occurrence of a piece of text or formatting in the document using a

While...Wend loop, remember to use **StartOfDocument** before the first **EditFind** instruction. You should either omit the .Wrap argument or set it to 0 (zero). If you set .Wrap to 1, you may create an endless loop. If you set .Wrap to 2, Word displays a message asking if you'd like to continue from the beginning of the document; you must choose No to keep the macro running.

See also

Finding and Replacing Statements and Functions

EditFindClearFormatting

EditFindFont

EditFindFound()

EditFindLang

EditFindPara

EditFindStyle

EditReplace

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditFindClearFormatting Example

The **EditFindClearFormatting** statement in this example ensures that the only formats specified are bold and italic:

```
EditFindClearFormatting
EditFindFont .Bold = 1, .Italic = 1
EditFind .Find = "", .Format = 1
```

EditFindClearFormatting

¬Example

EditFindClearFormatting

Clears any formats specified for text to be found. Before specifying formats with **EditFindFont**, **EditFindPara**, and so on, it's a good idea to include the **EditFindClearFormatting** statement to ensure that any unwanted formats are not included in the specification.

See also

Finding and Replacing Statements and Functions

EditFind

EditFindFont

EditFindLang

EditFindPara

EditFindStyle

EditReplace

EditReplaceClearFormatting

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditFindFont Examples

This example finds the next instance of any text that is bold:

```
EditFindClearFormatting
EditFindFont .Bold = 1
EditFind .Find = "", .Format = 1
```

The following example finds the next instance of the word "Note" that is not bold:

```
EditFindClearFormatting
EditFindFont .Bold = 0
EditFind .Find = "Note", .Format = 1
```

The following example finds the next instance of "Note" whether or not it is bold. Normally, you use **EditFindClearFormatting** to remove formatting specifications from a search operation. With the instruction `EditFindFont .Bold = -1`, you can remove the specification for bold without affecting other search formats set previously.

```
EditFindFont .Bold = -1
EditFind .Find = "Note", .Format = 1
```

EditFindFont

¬Example

EditFindFont [**.Points** = *number*] [**, .Underline** = *number*] [**, .Color** = *number*] [**, .Strikethrough** = *number*] [**, .Superscript** = *number*] [**, .Subscript** = *number*] [**, .Hidden** = *number*] [**, .SmallCaps** = *number*] [**, .AllCaps** = *number*] [**, .Spacing** = *number*] [**, .Position** = *number or text*] [**, .Kerning** = *number*] [**, .KerningMin** = *number or text*] [**, .Default**] [**, .Tab** = *number*] [**, .Font** = *text*] [**, .Bold** = *number*] [**, .Italic** = *number*]

When followed by an **EditFind** or **EditReplace** instruction in which .Format = 1, specifies the character formatting of the text you want to find. You can set .Strikethrough, .Superscript, .Subscript, .Hidden, .SmallCaps, .AllCaps, .Kerning, .Bold, and .Italic to one of the three following values.

Use this value	To do this
-1	Find text regardless of whether it has the format
0 (zero)	Find text that does not have the format
1	Find text that has the format

For more information on the arguments, see FormatFont.

See also

Finding and Replacing Statements and Functions

EditFind

EditFindClearFormatting

EditFindLang

EditFindPara

EditFindStyle

EditReplace

EditReplaceFont

FormatFont

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditFindFound() Example

This macro counts the number of times "pillbox" occurs in a document and then displays the result in a message box. **EditFindFound()** is often used with a **While...Wend** loop to repeat a series of instructions each time the specified text or formatting is found.

```
Sub MAIN
count = 0
StartOfDocument
EditFind .Find = "pillbox", .WholeWord = 1, .Format = 0, .Wrap = 0
While EditFindFound()
    count = count + 1
    EditFind
Wend
MsgBox "The word " + Chr$(34) + "pillbox" + Chr$(34) + \
    " was found" + Str$(count) + " times."
End Sub
```

EditFindFound()

¬Example

EditFindFound()

Returns a value indicating whether the most recent **EditFind** operation was successful.

Value	Explanation
-1	The find operation was successful.
0 (zero)	The find operation was not successful.

See also

Finding and Replacing Statements and Functions

EditFind

EditReplace

While...Wend

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditFindLang Example

This example formats all occurrences of German text with italic. Note that the instruction EditReplaceLang .Language = "Deutsch" is required to preserve the language formatting of the found text.

```
EditFindClearFormatting
EditReplaceClearFormatting
EditFindLang .Language = "Deutsch"
EditReplaceLang .Language = "Deutsch"
EditReplaceFont .Italic = 1
EditReplace .Find = "", .Replace = "", .Format = 1, \
    .ReplaceAll, .Wrap = 1
```

EditFindLang

¬Example

EditFindLang .Language = *text*

When followed by an **EditFind** or **EditReplace** instruction in which .Format is set to 1, specifies the language formatting of the text you want to find. Although the Language command (Tools menu) lists the names of languages in their English form, the text for .Language must be in the specified language and include any accented characters (for example, Italian must be specified as "Italiano" and French must be specified as "Français"). For a list of valid foreign language names, see ToolsLanguage.

See also

Finding and Replacing Statements and Functions

EditFind

EditFindFont

EditFindPara

EditFindStyle

EditReplace

EditReplaceLang

Language

ToolsLanguage

EditFindPara

EditFindPara [**.LeftIndent** = *number or text*] [**, .RightIndent** = *number or text*] [**, .Before** = *number or text*] [**, .After** = *number or text*] [**, .LineSpacingRule** = *number*] [**, .LineSpacing** = *number or text*] [**, .Alignment** = *number*] [**, .WidowControl** = *number*] [**, .KeepWithNext** = *number*] [**, .KeepTogether** = *number*] [**, .PageBreak** = *number*] [**, .NoLineNum** = *number*] [**, .DontHyphen** = *number*] [**, .Tab** = *number*] [**, .FirstIndent** = *number or text*]

When followed by an **EditFind** or **EditReplace** instruction in which .Format is set to 1, specifies the paragraph formatting of the text you want to find. You can set .WidowControl, .KeepWithNext, .KeepTogether, .PageBreak, .NoLineNum, and .DontHyphen to one of the three following values.

Use this value	To do this
-1	Find text regardless of whether it has the format
0 (zero)	Find text that does not have the format
1	Find text that has the format

For more information on the arguments, see FormatParagraph. For an example, see EditReplacePara Example.

See also

Finding and Replacing Statements and Functions

EditFind

EditFindFont

EditFindLang

EditFindStyle

EditReplace

EditReplacePara

FormatParagraph

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditFindStyle Examples

This example finds and selects the next paragraph formatted with the Heading 2 style:

```
EditFindClearFormatting
EditFindStyle .Style = "Heading 2"
EditFind .Find = "", .Format = 1
```

The following example finds the next instance of the word "catnip" in a paragraph of any style. Normally, you use **EditFindClearFormatting** to remove formatting specifications from a search operation. With the instruction `EditFindStyle .Style = ""`, you can remove the style specification without affecting other search formats set previously.

```
EditFindStyle .Style = ""
EditFind .Find = "catnip", .Format = 1
```

EditFindStyle

¬Example

EditFindStyle .Style = *text*

When followed by an **EditFind** or **EditReplace** instruction in which .Format is set to 1, specifies the style of the text you want to find. If the specified style does not exist in the active document, or if the capitalization does not match that of the actual style name, an error occurs.

Argument	Explanation
.Style	The name of the style you want to find; to remove a style from the specification, use an empty string ("").

See also

Finding and Replacing Statements and Functions

EditFind

EditFindFont

EditFindLang

EditFindPara

EditReplace

EditReplaceStyle

FormatStyle

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditReplace Example

This example finds all instances of underlined text in the document and replaces the format with italic. Note that the instruction .Wrap = 1 means no **StartOfDocument** statement is required; instances both before and after the insertion point are replaced.

```
EditFindClearFormatting
EditReplaceClearFormatting
EditFindFont .Underline = 1
EditReplaceFont .Italic = 1, .Underline = 0
EditReplace .Find = "", .Replace = "", .Format = 1, \
    .ReplaceAll, .Wrap = 1
```

EditReplace

¬Example

EditReplace [**.Find** = *text*] [**, .Replace** = *text*] [**, .Direction** = *number*] [**, .MatchCase** = *number*] [**, .WholeWord** = *number*] [**, .PatternMatch** = *number*] [**, .SoundsLike** = *number*] [**, .FindNext**] [**, .ReplaceOne**] [**, .ReplaceAll**] [**, .Format** = *number*] [**, .Wrap** = *number*]

Replaces one or all instances of the specified text or formatting (or both) with different text or formatting. If there is a selection and .Wrap is omitted or set to 0 (zero), the .Find text is replaced only within the selection. The arguments for the **EditReplace** statement correspond to the options in the Replace dialog box (Edit menu).

Argument	Explanation
.Find	The text to find, or, to search for formatting only, an empty string (""). You can search for special characters, such as paragraph marks, by specifying appropriate character codes. For example, "^p" corresponds to a paragraph mark and "^t" corresponds to a tab character. For a complete list of special characters, see Finding and replacing special characters by using keyboard codes. If .PatternMatch is set to 1, you can specify wildcard characters and other advanced search criteria. For example, "*(ing)" finds any word ending in "ing." For more information, see Advanced Search Criteria To search for a symbol character, specify the symbol's ANSI value following a caret (^) and a zero (0). For example, "^0151" corresponds to an em dash (---). For a table of symbols and their ANSI values, see Appendix A, "Character Sets," in the *Microsoft Word User's Guide*.
.Replace	The replacement text, or, to delete the text specified with .Find, an empty string (""). You specify special characters and advanced search criteria just as you do for .Find. To specify a graphic or other nontext item as the replacement, put the item on the Clipboard and specify "^c" for .Replace.
.Direction	The direction to search: 0 (zero) Searches toward the end of the document. 1 Searches toward the beginning of the document.

	The default is the direction used in the previous search or 0 (zero) the first time the Find or Replace command is run.
.MatchCase	If 1, corresponds to selecting the Match Case check box.
.WholeWord	If 1, corresponds to selecting the Find Whole Words Only check box.
.PatternMatch	If 1, Word evaluates .Find as a string containing advanced search criteria such as the asterisk (*) and question mark (?) wildcard characters (corresponds to selecting the Use Pattern Matching check box).
.SoundsLike	If 1, corresponds to selecting the Sounds Like check box.
.FindNext	Finds the next instance of the text specified by .Find.
.ReplaceOne	Replaces the first instance of the text specified by .Find.
.ReplaceAll	Replaces all instances of the .Find text.
.Format	Finds and replaces formatting in addition to, or instead of, text: 0 (zero) Ignores formatting. 1 Uses the specified formatting.
.Wrap	Controls what happens if the search begins at a point other than the beginning of the document and the end of the document is reached (or vice versa if .Direction is set to 1). The .Wrap argument also controls what happens if there is a selection and the search text is not found in the selection. 0 (zero) The replace operation ends and the macro continues. 1 If there is a selection, the replace operation continues in the remainder of the document. If the beginning or end of the document is reached, the replace operation continues from the opposite end. 2 If there is a selection, Word displays a message asking whether to continue the replace operation in the remainder of the document. If the beginning or end of the document is reached, Word displays a message asking whether to continue the replace operation from the opposite end.

Formatting is not specified within the **EditReplace** instruction itself. Instead, you precede **EditReplace** with one or more of the following statements: **EditFindFont**, **EditFindLang**, **EditFindPara**, **EditFindStyle**, **EditReplaceFont**, **EditReplaceLang**, **EditReplacePara**, or **EditReplaceStyle**. Then, in the **EditReplace** instruction, you set .Format to 1.

Here are some tips for using **EditReplace** effectively:

- **EditReplace** retains whatever settings were used in the most recent find or replace operation. For example, if the Find Whole Words Only check box was selected in the Replace dialog box in the last replace operation (or if the .WholeWord argument was set to 1), .WholeWord will automatically be set to 1 in the new replace operation---unless you explicitly set it to 0 (zero). Before beginning a replace operation, make sure to specify every option that can affect the outcome of a search.
- When you search for and replace formatting, it's a good idea to include the statements **EditFindClearFormatting** and **EditReplaceClearFormatting** before specifying the formats you want to find and replace. That way, you clear any formats left over from previous find or replace operations.

See also

Finding and Replacing Statements and Functions

EditFind

EditReplaceClearFormatting

EditReplaceFont

EditReplaceLang

EditReplacePara

EditReplaceStyle

EditReplaceClearFormatting

EditReplaceClearFormatting

Clears any formats specified for replacement text. Before specifying replacement formats with **EditReplaceFont**, **EditReplacePara**, and so on, it's a good idea to include the **EditReplaceClearFormatting** statement to ensure that any unwanted formats are not included in the replacement formatting. For an example, see EditReplace Example.

See also

Finding and Replacing Statements and Functions

EditFindClearFormatting

EditReplace

EditReplaceFont

EditReplaceLang

EditReplacePara

EditReplaceStyle

EditReplaceFont

EditReplaceFont [**.Points** = *number*] [**, .Underline** = *number*] [**, .Color** = *number*] [**, .Strikethrough** = *number*] [**, .Superscript** = *number*] [**, .Subscript** = *number*] [**, .Hidden** = *number*] [**, .SmallCaps** = *number*] [**, .AllCaps** = *number*] [**, .Spacing** = *number*] [**, .Position** = *number or text*] [**, .Kerning** = *number*] [**, .KerningMin** = *number or text*] [**, .Default**] [**, .Tab** = *number*] [**, .Font** = *text*] [**, .Bold** = *number*] [**, .Italic** = *number*]

When followed by an **EditReplace** instruction in which .Format is set to 1, specifies the character formatting of the replacement text. You can
set .Strikethrough, .Superscript, .Subscript, .Hidden, .SmallCaps, .AllCaps, .Kerning, .Bold, and .Italic to one of the three following values.

Use this value	To do this
-1	Preserve the state of a given format in the found text
0 (zero)	Remove the given format from the found text
1	Apply the given format to the found text

For more information on the arguments, see FormatFont. For an example, see EditFindLang Example.

See also

Finding and Replacing Statements and Functions

EditFindFont

EditReplace

EditReplaceClearFormatting

EditReplaceLang

EditReplacePara

EditReplaceStyle

EditReplaceLang

EditReplaceLang .Language = *text*

When followed by an **EditReplace** instruction in which .Format is set to 1, specifies the language formatting of the replacement text. Although the Language command (Tools menu) lists the names of languages in their English form, the text for .Language must be in the specified language and include any accented characters (for example, Italian must be specified as "Italiano" and French must be specified as "Français"). For a list of valid foreign language names, see ToolsLanguage.

For an example, see EditFindLang Example.

_____

See also

Finding and Replacing Statements and Functions

EditFindLang

EditReplace

EditReplaceClearFormatting

EditReplaceFont

EditReplacePara

EditReplaceStyle

ToolsLanguage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditReplacePara Example

This example applies a 2-inch left indent to all paragraphs in the active document that currently have a 1-inch left indent:

```
StartOfDocument
EditFindClearFormatting
EditReplaceClearFormatting
EditFindPara .LeftIndent = "1 in"
EditReplacePara .LeftIndent = "2 in"
EditReplace .Find = "", .Replace = "", .ReplaceAll, \
    .Format = 1, .Wrap = 0
```

EditReplacePara

¬Example

EditReplacePara [**.LeftIndent** = *number or text*] [**, .RightIndent** = *number or text*] [**, .Before** = *number or text*] [**, .After** = *number or text*] [**, .LineSpacingRule** = *number*] [**, .LineSpacing** = *number or text*] [**, .Alignment** = *number*] [**, .WidowControl** = *number*] [**, .KeepWithNext** = *number*] [**, .KeepTogether** = *number*] [**, .PageBreak** = *number*] [**, .NoLineNum** = *number*] [**, .DontHyphen** = *number*] [**, .Tab** = *number*] [**, .FirstIndent** = *number or text*]

When followed by an **EditReplace** instruction in which .Format is set to 1, specifies paragraph formatting for the replacement text. You can set .WidowControl, .KeepWithNext, .KeepTogether, .PageBreak, .NoLineNum, and .DontHyphen to one of the three following values.

Use this value	To do this
-1	Preserve the state of a given format in the found text
0 (zero)	Remove the given format from the found text
1	Apply the given format to the found text

For more information on the arguments, see FormatParagraph.

See also

Finding and Replacing Statements and Functions

EditFindPara

EditReplace

EditReplaceClearFormatting

EditReplaceFont

EditReplaceLang

EditReplaceStyle

FormatParagraph

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditReplaceStyle Example

This example changes all Heading 4 paragraphs to Heading 3 paragraphs:

```
StartOfDocument
EditFindClearFormatting
EditReplaceClearFormatting
EditFindStyle .Style = "Heading 4"
EditReplaceStyle .Style = "Heading 3"
EditReplace .Find = "", .Replace = "", .ReplaceAll, \
    .Format = 1, .Wrap = 0
```

EditReplaceStyle

¬Example

EditReplaceStyle .Style = *text*

When followed by an **EditReplace** instruction in which .Format is set to 1, specifies a style for the replacement text. If the specified style does not exist in the active document, or if the capitalization does not match that of the actual style name, an error occurs.

Argument	Explanation
.Style	The name of the style for the replacement text; to remove a style from the replacement formatting, use an empty string ("").

See also

Finding and Replacing Statements and Functions

EditFindStyle

EditReplace

EditReplaceClearFormatting

EditReplaceFont

EditReplaceLang

EditReplacePara

FormatStyle

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

RepeatFind Example

This example counts the number of times the word "success" occurs in the active document and then displays the result in a message box:

```
StartOfDocument
EditFind .Find = "success", .Direction = 0, .WholeWord = 1, \
    .MatchCase = 0, .Format = 0, .Wrap = 0
While EditFindFound()
    count = count + 1
    RepeatFind
Wend
MsgBox "The word " + Chr$(34) + "success" + Chr$(34) \
    + " occurs" + Str$(Count) + " times."
```

RepeatFind

¬Example

RepeatFind

Repeats the most recent **EditFind** or **EditGoTo** operation. Note that when an **EditFind** instruction is repeated, settings for .Direction, .WholeWord, .MatchCase, .PatternMatch, .Format, and .Wrap are all repeated.

_____

See also

Finding and Replacing Statements and Functions

EditFind

EditGoTo

EditRepeat

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

AnnotationRefFromSel$() Example

This example goes to the next annotation mark and sets the variable mark$ to the characters in the annotation mark:

```
EditGoTo "a"
mark$ = AnnotationRefFromSel$()
```

AnnotationRefFromSel$()

¬Example

AnnotationRefFromSel$()

Returns the annotation mark associated with the insertion point or the beginning of the selection. The insertion point must be immediately before an annotation mark in the document window or within an annotation in the annotation pane; otherwise, the characters "[0]" are returned.

Note that because Word treats an entire annotation mark as an individual character, you cannot return the characters within an annotation mark using the **Selection$()** function.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

EditGoTo

GoToAnnotationScope

GoToNext*Item*

GoToPrevious*Item*

ShowAnnotationBy

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditConvertAllEndnotes Example

This example converts all endnotes to footnotes. If WordBasic error 509 ("This command is not available") occurs, the error handler displays the message "No endnotes to convert" instead of the standard WordBasic error message. Other error conditions generate standard messages.

```
On Error Goto trap
EditConvertAllEndnotes
trap:
Select Case Err
    Case 509 : MsgBox "No endnotes to convert."
    Case Else : Error Err
End Select
```

EditConvertAllEndnotes

¬Example

EditConvertAllEndnotes

Converts all endnotes in the active document to footnotes. An error occurs if there are no endnotes.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

EditConvertAllFootnotes

EditConvertNotes

EditSwapAllNotes

EditConvertAllFootnotes

EditConvertAllFootnotes

Converts all footnotes in the active document to endnotes. An error occurs if there are no footnotes. For an example using a similar statement, see EditConvertAllEndnotes Example.

See also

Footnotes, Endnotes, and Annotations Statements and Functions

EditConvertAllEndnotes

EditConvertNotes

EditSwapAllNotes

EditConvertNotes

EditConvertNotes

Converts the selected footnotes to endnotes, or vice versa. An error occurs if the insertion point or selection is not in the footnote or endnote pane, or if there are no notes in the pane.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

EditConvertAllEndnotes

EditConvertAllFootnotes

EditSwapAllNotes

EditSwapAllNotes

EditSwapAllNotes

Converts all footnotes in the document to endnotes and all endnotes to footnotes. The insertion point may be in either the document window or the footnote or endnote pane. If there are no footnotes or endnotes, an error occurs.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

EditConvertAllEndnotes

EditConvertAllFootnotes

EditConvertNotes

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GoToAnnotationScope Example

This example copies to the Clipboard the document text associated with the annotation containing the insertion point. The first **If** instruction ensures that the annotation pane is open and the insertion point is within it. Then **GoToAnnotationScope** selects the associated document text. The second **If** instruction ensures that text is selected before running **EditCopy**.

```
If ViewAnnotations() = -1 And WindowPane() = 3 Then
    GoToAnnotationScope
    If SelType() = 2 Then
        EditCopy
    Else
        MsgBox "No text to select."
    End If
    OtherPane
Else
    MsgBox "Please place the insertion point in an annotation."
End If
```

GoToAnnotationScope

¬Example

GoToAnnotationScope

Selects the range of document text associated with the annotation containing the insertion point. If the insertion point is not within an annotation in the annotation pane, an error occurs.

The range, or "scope," of the annotation is defined as the text that was selected when the annotation mark was inserted. This text appears shaded when the insertion point is within the annotation in the annotation pane. If no text was selected when the annotation was inserted, **GoToAnnotationScope** moves the insertion point to the position immediately preceding the annotation mark in the document text.

See also

Footnotes, Endnotes, and Annotations Statements and Functions

GoToNext*Item*

GoToPrevious*Item*

OtherPane

ViewAnnotations()

WindowPane()

InsertAnnotation

InsertAnnotation

Inserts an annotation mark at the insertion point, opens the annotation pane, and places the insertion point in the annotation pane. If text is selected, the annotation mark is inserted immediately after the selected text, which appears shaded when the insertion point is in the associated annotation in the annotation pane. If the document window is too small to display the annotation pane, an error occurs.

See also

Footnotes, Endnotes, and Annotations Statements and Functions

GoToAnnotationScope

InsertFootnote

ShowAnnotationBy

ViewAnnotations

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertFootnote Example

This example inserts an automatically numbered footnote reference mark at the insertion point, inserts a footnote, and then closes the footnote pane:

```
InsertFootnote .NoteType = 0
Insert "Sara Levine, "
Italic 1 : Insert "The Willow Tree"
Italic 0 : Insert " (Lone Creek Press, 1993)."
ClosePane
```

InsertFootnote

¬Example

InsertFootnote [**.Reference =** *text*] [**, .NoteType =** *number*]

Inserts a footnote or endnote reference mark at the insertion point, opens the footnote or endnote pane, and places the insertion point in the pane. The arguments for the **InsertFootnote** statement correspond to the options in the Footnote And Endnote dialog box (Footnote command, Insert menu).

Argument	Explanation
.Reference	A custom reference mark. If omitted, Word inserts an automatically numbered reference mark. To specify a symbol, use the syntax **{***FontName CharNum***}**. *FontName* is the name of a font containing the symbol. Names of decorative fonts appear in the Font box in the Symbol dialog box (Insert menu). *CharNum* is the sum of 31 and the number corresponding to the position of the symbol you want to insert, counting from left to right in the table of symbols. You can also look up character codes in Appendix A, "Character Sets," in the *Microsoft Word User's Guide*. For example, to specify an omega (Ω) at position 56 in the table of symbols in the Symbol font, include the argument .Reference = "{Symbol 87}".
.NoteType	Specifies whether to insert a footnote or endnote: 0 (zero) Footnote 1 Endnote Omitted The type that was inserted most recently

See also

Footnotes, Endnotes, and Annotations Statements and Functions

InsertAnnotation

NoteOptions

ViewFootnoteArea

ViewFootnotes

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

NoteOptions Example

This example places footnotes beneath text, using the "A, B, C" format and starting with "A"; footnote numbers restart with each new section:

```
NoteOptions .FootnotesAt = 1, .FootNumberAs = 2, \
        .FootStartingNum = "1", .FootRestartNum = 1
```

NoteOptions

¬Example

NoteOptions [**.FootnotesAt =** *number*] [**, .FootNumberAs** = *number*] [**, .FootStartingNum =** *text*] [**, .FootRestartNum** = *number*] [**, .EndnotesAt =** *number*] [**, .EndNumberAs =** *number*] [**, .EndStartingNum =** *text*] [**, .EndRestartNum =** *number*]

Specifies the placement and formatting of footnotes and endnotes. The arguments for the **NoteOptions** statement correspond to the options in the Note Options dialog box (Footnote command, Insert menu).

Argument	Explanation
.FootnotesAt	Specifies where to place footnotes: 0 (zero) Bottom Of Page 1 Beneath Text
.FootNumberAs	Specifies the format for footnote reference marks: 0 (zero) 1, 2, 3,... 1 a, b, c,… 2 A, B, C,... 3 i, ii, iii,... 4 I, II, III,... 5 *, †, ‡, §,...
.FootStartingNum	The starting number for footnotes; if .FootStartingNum is not 1, .FootRestartNum must be set to 0 (zero).
.FootRestartNum	Specifies how to number footnotes after page breaks or section breaks: 0 (zero) Continuous 1 Restart Each Section 2 Restart Each Page
.EndnotesAt	Specifies where to place endnotes: 0 (zero) End Of Section 1 End Of Document
.EndNumberAs	Specifies the format for endnote reference marks. For values, see the .FootNumberAs argument.
.EndStartingNum	The starting number for endnotes; if .EndStartingNum is not 1, .EndRestartNum must be set to 0 (zero).
.EndRestartNum	Specifies how to number endnotes after section breaks: 0 (zero) Continuous 1 Restart Each Section

See also

Footnotes, Endnotes, and Annotations Statements and Functions

InsertFootnote

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ResetNoteSepOrNotice Example

This example opens the footnote continuation notice pane and then determines if there is a continuation notice. If the pane contains anything more than a paragraph mark, ResetNoteSepOrNotice removes the notice. Without the **If** conditional, the example would generate an error if the notice were already set to the default.

```
ViewFootnoteContNotice
EditSelectAll
paraMark$ = Chr$(13)
If Selection$() <> paraMark$ Then ResetNoteSepOrNotice
ClosePane
```

ResetNoteSepOrNotice

¬Example

ResetNoteSepOrNotice

Resets the separator, the continuation notice, or the continuation separator for footnotes or endnotes to the default. **ResetNoteSepOrNotice** generates an error unless you first open a separator pane.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

NoteOptions

ViewEndnoteContNotice

ViewEndnoteContSeparator

ViewEndnoteSeparator

ViewFootnoteContNotice

ViewFootnoteContSeparator

ViewFootnoteSeparator

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ShowAnnotationBy Example

This example instructs Word to display annotations by Sara Levine only, and then displays the annotation pane:

```
ShowAnnotationBy "Sara Levine"
ViewAnnotations 1
```

ShowAnnotationBy

¬Example

ShowAnnotationBy *ReviewerName$*

Displays the annotations by *ReviewerName$* when the annotation pane is open. *ReviewerName$* must match a reviewer in the list at the top of the annotation pane. To display all annotations, specify "All Reviewers" as *ReviewerName$*.

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ViewAnnotations

ViewAnnotations, ViewAnnotations()

ViewAnnotations [*On*]

ViewAnnotations()

The **ViewAnnotations** statement opens or closes the annotation pane. An error occurs if there are no annotations in the active document.

Argument	Explanation
On	Specifies whether to open or close the annotation pane: 0 (zero) Closes the annotation pane. 1 Opens the annotation pane. Omitted Toggles the display of the annotation pane.

The **ViewAnnotations()** function returns the following values.

Value	Explanation
0 (zero)	If the annotation pane is closed
-1	If the annotation pane is open

See also

Footnotes, Endnotes, and Annotations Statements and Functions

InsertAnnotation

ViewEndnoteArea

ViewFootnoteArea

ViewFootnotes

ViewEndnoteArea, ViewEndnoteArea()

ViewEndnoteArea [*On*]

ViewEndnoteArea()

In normal and outline views, the **ViewEndnoteArea** statement opens or closes the endnote pane. In page layout view, **ViewEndnoteArea** moves the insertion point to or from the endnote area. If there are no endnotes in the active document, an error occurs.

Argument	Explanation
On	Specifies whether to open or close the endnote pane: 0 (zero) In normal and outline views, closes the endnote pane; in page layout view, has no effect. 1 In normal and outline views, opens the endnote pane; in page layout view, moves the insertion point between an endnote and its associated reference mark in the document text. Omitted In normal and outline views, toggles the display of the endnote pane; in page layout view, moves the insertion point between an endnote and its associated reference mark in the document text.

The **ViewEndnoteArea()** function returns the following values.

Value	Explanation
0 (zero)	If the endnote pane is closed
-1	If the endnote pane is open

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ViewAnnotations

ViewFootnoteArea

ViewFootnotes

ViewEndnoteContNotice

ViewEndnoteContNotice

Opens a pane containing the endnote continuation notice, which indicates that an endnote is continued on the following page.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ResetNoteSepOrNotice

ViewEndnoteContSeparator

ViewEndnoteSeparator

ViewFootnoteContNotice

ViewEndnoteContSeparator

ViewEndnoteContSeparator

Opens a pane containing the endnote continuation separator, which appears before endnote text that is continued from the previous page.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ResetNoteSepOrNotice

ViewEndnoteContNotice

ViewEndnoteSeparator

ViewFootnoteContSeparator

ViewEndnoteSeparator

ViewEndnoteSeparator

Opens a pane containing the endnote separator, which appears between document text and the endnotes.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ResetNoteSepOrNotice

ViewEndnoteContNotice

ViewEndnoteContSeparator

ViewFootnoteSeparator

ViewFootnoteContNotice

ViewFootnoteContNotice

Opens a pane containing the footnote continuation notice, which indicates that a footnote is continued on the following page.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ResetNoteSepOrNotice

ViewEndnoteContNotice

ViewFootnoteContSeparator

ViewFootnoteSeparator

ViewFootnoteContSeparator

ViewFootnoteContSeparator

Opens a pane containing the footnote continuation separator, which appears before footnote text that is continued from the previous page.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ResetNoteSepOrNotice

ViewEndnoteContSeparator

ViewFootnoteContNotice

ViewFootnoteSeparator

ViewFootnotes, ViewFootnotes()

ViewFootnotes

ViewFootnotes()

In normal view, the **ViewFootnotes** statement opens or closes the footnote pane or endnote pane according to the following rules:

- If the document contains footnotes, opens the footnote pane.
- If the document contains endnotes, but no footnotes, opens the endnote pane.
- If the footnote pane or endnote pane is already open, closes the pane.

In page layout view, **ViewFootnotes** moves the insertion point according to the following rules:

- If the document has only footnotes, moves the insertion point to the footnote area; if it has only endnotes, moves the insertion point to the endnote area.
- If the document has both footnotes and endnotes, displays a dialog box that asks the user to choose between the footnote and endnote area.
- If the insertion point is already in a footnote or endnote area, moves the insertion point to the document area.

If the document contains no footnotes or endnotes, **ViewFootnotes** has no effect.

The **ViewFootnotes()** function returns the following values.

Value	Explanation
0 (zero)	If neither the footnote nor endnote pane is open
-1	If either the footnote or endnote pane is open

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ViewEndnoteArea

ViewFootnoteArea

ViewFootnoteSeparator

ViewFootnoteSeparator

Opens a pane containing the footnote separator, which appears between document text and the footnotes.

_____

See also

Footnotes, Endnotes, and Annotations Statements and Functions

ResetNoteSepOrNotice

ViewEndnoteSeparator

ViewFootnoteContNotice

ViewFootnoteContSeparator

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

AddDropDownItem Example

This example adds three items to the drop-down form field marked by the bookmark "Dropdown1":

```
AddDropDownItem "Dropdown1", "Red"
AddDropDownItem "Dropdown1", "Blue"
AddDropDownItem "Dropdown1", "Green"
```

AddDropDownItem

¬Example

AddDropDownItem *BookmarkName$*, *ItemText$*

Adds an item to a drop-down form field. A drop-down form field can hold as many as 25 items; if you attempt to add more than 25 items, an error occurs.

Argument	Explanation
BookmarkName$	The name of the bookmark that marks the drop-down form field. If you specify a bookmark that does not mark a drop-down form field, an error occurs.
ItemText$	The item to add to the drop-down form field.

See also

Forms Statements and Functions

DropDownFormField

RemoveAllDropDownItems

RemoveDropDownItem

CheckBoxFormField

CheckBoxFormField

Inserts a check box form field at the insertion point. **CheckBoxFormField** corresponds to the Check Box Form Field button on the Forms toolbar. For information about using form fields, see Chapter 14, "Forms," in the *Microsoft Word User's Guide*.

_____

See also

Forms Statements and Functions

DropDownFormField

InsertFormField

TextFormField

DropDownFormField

DropDownFormField

Inserts a drop-down form field at the insertion point. **DropDownFormField** corresponds to the Drop-Down Form Field button on the Forms toolbar. For information about using form fields, see Chapter 14, "Forms," in the *Microsoft Word User's Guide*.

_____

See also

Forms Statements and Functions

AddDropDownItem

CheckBoxFormField

InsertFormField

RemoveAllDropDownItems

RemoveDropDownItem

TextFormField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EnableFormField Example

This example allows or prevents changes to a form field based on the user name specified on the User Info tab in the Options dialog box (Tools menu). This macro could be part of an AutoNew macro that runs each time a document based on the form template is created.

```
Dim dlg As ToolsOptionsUserInfo
GetCurValues dlg
If dlg.Name = "Stella Richards" Then
    EnableFormField "TotalSales", 1
Else
    EnableFormField "TotalSales", 0
End If
```

EnableFormField

¬Example

EnableFormField *BookmarkName$*, *Enable*

Allows or prevents changes to the specified form field while the form is being filled in.

Argument	Explanation
BookmarkName$	The name of the bookmark that marks the form field.
Enable	Specifies whether the form field can be filled in or otherwise changed: 0 (zero) The form field cannot be changed. 1 The form field can be changed.

See also

Forms Statements and Functions

InsertFormField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormFieldOptions Examples

This example specifies help text for the selected form field. Help text is specified for both the status bar and the box that Word displays when the F1 key is pressed.

```
FormFieldOptions .OwnHelp = 1, .HelpText = "Briefly summarize " + \
    "your favorite activity. Provide specific details.", \
    .OwnStat = 1, .StatText = "What you like to do best..."
```

This example uses the **FormFieldOptions** dialog record to define the variable curType with a number that corresponds to the .Type setting for the selected form field. You can use this example as a model for how to return information on any of the options you can set with **FormFieldOptions**.

```
Dim dlg As FormFieldOptions
GetCurValues dlg
curType = dlg.Type
```

FormFieldOptions

¬Example

FormFieldOptions [**.Entry** = *text*] [**, .Exit** = *text*] [**, .Name** = *text*] [**, .Enable** = *number*] [**, .TextType** = *number*] [**, .TextWidth** = *number or text*] [**, .TextDefault** = *text*] [**, .TextFormat** = *text*] [**, .CheckSize** = *number*] [**, .CheckWidth** = *number or text*] [**, .CheckDefault** = *number*] [**, .Type** = *number*] [**, .OwnHelp** = *number*] [**, .HelpText** = *text*] [**, .OwnStat** = *number*] [**, .StatText** = *text*]

Changes the properties of a selected form field. The arguments for the **FormFieldOptions** statement correspond to the options in the dialog box that Word displays when you select a form field and click the Form Field Options button on the Forms toolbar or choose Form Field Options from the shortcut menu. If more than one form field is selected, **FormFieldOptions** changes the properties of the first form field in the selection. If no form field is selected, or if you attempt to set options that are inappropriate for the selected form field, an error occurs.

Argument	Explanation
.Entry	The macro that runs when the form field receives the focus.
.Exit	The macro that runs when the form field loses the focus.
.Name	The name of the bookmark that will mark the form field.
.Enable	If 1, allows the form field to be changed as the form is filled in.
.TextType	For a text form field, specifies the type: 0 (zero) Regular Text 1 Number 2 Date 3 Current Date 4 Current Time 5 Calculation
.TextWidth	For a text form field, specifies a maximum width: 0 (zero) Unlimited > 0 A maximum width, in characters
.TextDefault	The default text for a text form field.
.TextFormat	For a text form field, specifies a format appropriate for .TextType. If .TextType is set to 0 (zero) for Regular Text, the following values are available: Uppercase, Lowercase, First Capital, Title Case. For formats available to other text types, review the items in the Text Format box in the Text Form Field Options dialog box.
.CheckSize	For a check box form field, specifies whether to set a fixed size: 0 (zero) Auto (the size is determined by the font size of the surrounding text)

	1 Exactly (the size is fixed to that specified by .CheckWidth)
.CheckWidth	For a check box form field, and if .CheckSize is set to 1, specifies a fixed width and height in points or a text measurement.
.CheckDefault	For a check box form field, specifies whether the check box should be cleared or selected by default: 0 (zero) Cleared 1 Selected
.Type	Specifies the type of form field: 0 (zero) or omitted Text form field 1 Check box form field 2 Drop-down form field
.OwnHelp	Specifies the source of the text that is displayed in a message box when the form field has the focus and the F1 key is pressed: 0 (zero) The text of the AutoText entry specified by .HelpText. If .HelpText specifies an empty string (""), no message box is displayed 1 The text specified by .HelpText
.HelpText	If .OwnHelp is set to 0, the name of an AutoText entry containing help text for the form field. If .OwnHelp is set to 1, help text.
.OwnStat	Specifies the source of the text that is displayed in the status bar when the form field has the focus: 0 (zero) The text of the AutoText entry specified by .StatText. If .StatText specifies an empty string (""), no text is displayed 1 The text specified by .StatText
.StatText	If .OwnStat is set to 0, the name of an AutoText entry containing status bar text for the form field. If .OwnStat is set to 1, the text to be displayed in the status bar.

You can use the **FormFieldOptions** dialog record to retrieve information about the selected form field. For details, see the second example in this reference entry.

See also

Forms Statements and Functions

EnableFormField

InsertFormField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetFormResult() Example

This is an example of an "on-exit" macro that tests the item selected in a drop-down form field as soon as the user moves the focus to another form field. If the item is "Paris," a message box is displayed. The on-exit macro is assigned to the field marked by the bookmark "Dropdown1."

```
Sub MAIN
city$ = GetFormResult$("Dropdown1")
If city$ = "Paris" Then
    MsgBox "We offer daily excursions to Versailles!"
End If
End Sub
```

For more information about on-exit macros, see Chapter 9, "More WordBasic Techniques," in the Microsoft Word Developer's Kit. For further information about forms and form fields, see Chapter 14, "Forms," in the *Microsoft Word User's Guide*.

GetFormResult(), GetFormResult$()

¬Example

GetFormResult(*BookmarkName$***)**

GetFormResult$(***BookmarkName$*)**

GetFormResult() returns a number corresponding to the setting of a check box form field or drop-down form field marked by the bookmark *BookmarkName$*. (Word adds a bookmark automatically when you insert a form field.) **GetFormResult()** returns values as follows:

- For a check box form field, returns 0 (zero) if the check box is cleared or 1 if it is selected.
- For a drop-down form field, returns 0 (zero) if the first item is selected, 1 if the second item is selected, and so on.

Note that an error occurs if you use **GetFormResult()** with a text form field.

GetFormResult$() returns a string corresponding to the setting of the form field marked by the bookmark *BookmarkName$*. **GetFormResult$()** returns values as follows:

- For a check box form field, returns "0" (zero) if the check box is cleared or "1" if it is selected.
- For a drop-down form field, returns the item that is currently selected.
- For a text form field, returns the current text.

_____

See also

Forms Statements and Functions

SetFormResult

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertFormField Examples

This example inserts a drop-down form field for which status bar text is specified and then adds the names of three cities to the form field:

```
InsertFormField .Name = "Cities", .Enable = 1, .Type = 2, \
        .OwnStat = 1, .StatText = "Select a city."
AddDropDownItem "Cities", "Bonn"
AddDropDownItem "Cities", "Paris"
AddDropDownItem "Cities", "Tokyo"
```

The following example inserts a check box form field that is 8 points high and wide and is selected by default. WriteSettings is specified as the "on-exit" macro that runs when the check box loses the focus. In a macro, instructions following this one could use **GetFormResult()** and **SetPrivateProfileString** to record in a text file whether the user selects or clears the check box.

```
InsertFormField .Exit = "WriteSettings", .Name = "Check1", \
        .Enable = 1, .CheckSize = 1, .CheckWidth = "8 pt", \
        .CheckDefault = 1, .Type = 1
```

The following example inserts a text form field. The argument .TextType = 1 specifies that the text is a number, and .TextFormat = "0%" specifies that the number be shown as a percentage. The default percentage is 100 percent.

```
InsertFormField .Name = "PercentComplete", .Enable = 1, \
        .TextType = 1, .TextWidth = "4", \
        .TextDefault = "100%", .TextFormat = "0%", .Type = 0
```

InsertFormField

¬Example

InsertFormField [**.Entry** = *text*] [**, .Exit** = *text*] [**, .Name** = *text*] [**, .Enable** = *number*] [**, .TextType** = *number*] [**, .TextDefault** = *text*] [**, .TextWidth** = *number or text*] [**, .TextFormat** = *text*] [**, .CheckSize** = *number*] [**, .CheckWidth** = *number or text*] [**, .CheckDefault** = *number*] [**, .Type** = *number*] [**, .OwnHelp** = *number*] [**, .HelpText** = *text*] [**, .OwnStat** = *number*] [**, .StatText** = *text*]

Inserts a form field at the insertion point. The arguments for the **InsertFormField** statement correspond to the options in the Form Field (Insert menu) and Form Field Options dialog boxes.

Argument	Explanation
.Entry	The macro that runs when the form field receives the focus.
.Exit	The macro that runs when the form field loses the focus.
.Name	A name for a bookmark that will mark the form field.
.Enable	If 1, allows the form field to be changed as the form is filled in.
.TextType	When inserting a text form field, specifies the type: 0 (zero) Regular Text 1 Number 2 Date 3 Current Date 4 Current Time 5 Calculation
.TextDefault	The default text for a text form field.
.TextWidth	When inserting a text form field, specifies a maximum width: 0 (zero) Unlimited > 0 A maximum width, in characters
.TextFormat	When inserting a text form field, specifies a format appropriate for .TextType. If .TextType is set to 0 (zero) for Regular Text, the following values are available: Uppercase, Lowercase, First Capital, Title Case. For formats available to other text types, review the items in the Text Format box in the Text Form Field Options dialog box.
.CheckSize	When inserting a check box form field, specifies whether to set a fixed size: 0 (zero) Auto (the size is determined by the font size of the surrounding text). 1 Exactly (the size is fixed to that specified by .CheckWidth).

.CheckWidth	When inserting a check box form field, and if .CheckSize is set to 1, specifies a fixed width and height in points or a text measurement.
.CheckDefault	When inserting a check box form field, specifies whether the check box should be cleared or selected by default: 0 (zero) Cleared 1 Selected
.Type	Specifies the type of form field to insert: 0 (zero) or omitted Text form field 1 Check box form field 2 Drop-down form field
.OwnHelp	Specifies the source of the text that is displayed in a message box when the form field has the focus and the F1 key is pressed: 0 (zero) The text of the AutoText entry specified by .HelpText. If .HelpText specifies an empty string (""), no message box is displayed 1 The text specified by .HelpText
.HelpText	If .OwnHelp is set to 0, the name of an AutoText entry containing help text for the form field. If .OwnHelp is set to 1, help text.
.OwnStat	Specifies the source of the text that is displayed in the status bar when the form field has the focus: 0 (zero) The text of the AutoText entry specified by .StatText. If .StatText specifies an empty string (""), no text is displayed 1 The text specified by .StatText
.StatText	If .OwnStat is set to 0, the name of an AutoText entry containing status bar text for the form field. If .OwnStat is set to 1, the text to be displayed in the status bar.

See also

Forms Statements and Functions

AddDropDownItem

CheckBoxFormField

DropDownFormField

EnableFormField

FormFieldOptions

RemoveAllDropDownItems

RemoveDropDownItem

TextFormField

RemoveAllDropDownItems

RemoveAllDropDownItems *BookmarkName$*

Removes all items from a drop-down form field.

Argument	Explanation
BookmarkName$	The name of the bookmark that marks the drop-down form field. If you specify a bookmark that does not mark a drop-down form field, an error occurs.

See also

Forms Statements and Functions

AddDropDownItem

DropDownFormField

RemoveDropDownItem

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

RemoveDropDownItem Example

This example removes three items from the drop-down form field marked by the bookmark "Dropdown1":

```
RemoveDropDownItem "Dropdown1", "Red"
RemoveDropDownItem "Dropdown1", "Blue"
RemoveDropDownItem "Dropdown1", "Green"
```

RemoveDropDownItem

¬Example

RemoveDropDownItem *BookmarkName$, ItemText$*

Removes an item from a drop-down form field.

Argument	Explanation
BookmarkName$	The name of the bookmark that marks the drop-down form field. If you specify a bookmark that does not mark a drop-down form field, an error occurs.
ItemText$	The item to remove from the drop-down form field.

See also

Forms Statements and Functions

AddDropDownItem

DropDownFormField

RemoveAllDropDownItems

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetFormResult Examples

The following example is an AutoNew macro that runs each time a form is created. It sets the default results for the text form field "NameText," the check box form field "AgeCheck," and the drop-down form field "PreferencesList." It then protects the document for forms.

```
Sub MAIN
SetFormResult "NameText", "John Jones", 1
SetFormResult "AgeCheck", 1, 1
SetFormResult "PreferencesList", 2, 1
ToolsProtectDocument .Type = 2
End Sub
```

The following example updates a text form field identified by the bookmark "DateField." The form field contains a date field, which is updated by the SetFormResult instruction.

```
SetFormResult "DateField"
```

SetFormResult

¬Example

SetFormResult *BookmarkName$***,** *Result*[*$*] [**,** *DefaultResult*]

Sets the result of the form field marked by the bookmark *BookmarkName$*. (Word adds a bookmark automatically when you insert a form field.) Depending on the type of form field, you specify *Result*[*$*] as a string or number, as follows:

- For a check box form field, specify 0 (zero) to clear the check box or 1 to select it.
- For a drop-down form field, specify 0 (zero) to select the first item, 1 to select the second item, and so on. You can also specify the item itself as a string.
- For a text form field, specify a string.

If you do not specify *Result*[*$*], **SetFormResult** updates any fields within the specified form field.

If *DefaultResult* is 1, the result becomes the default result for the field. The default result is displayed each time the document containing the form field is protected for forms.

_____

See also

Forms Statements and Functions

GetFormResult()

TextFormField

TextFormField

Inserts a text form field at the insertion point. **TextFormField** corresponds to the Text Form Field button on the Forms toolbar. For information about using form fields, see Chapter 14, "Forms," in the *Microsoft Word User's Guide*.

_____

See also

Forms Statements and Functions

CheckBoxFormField

DropDownFormField

InsertFormField

ToolsProtectDocument

ToolsProtectDocument [**.DocumentPassword =** *text*] [**, .NoReset =** *number*] [**, .Type =** *number*]

Protects the document from changes. According to the value of .Type, the user can make limited changes such as adding annotations or revision marks. If the document is already protected, an error occurs.

Argument	Explanation
.DocumentPassword	The password required to "unprotect" the document after choosing Unprotect Document (Tools menu).
.NoReset	If 1, Word does not reset form fields to their default results when a form is protected from changes using **ToolsProtectDocument** (applies only if .Type is 2). This option allows the form designer to create an "on entry" or "on exit" macro that removes protection from the form, modifies the form, and then protects it again without resetting form fields. Note that form fields are always reset when you protect a form using the Forms toolbar or the Protect Document dialog box (Tools menu).
.Type	The type of protection: 0 (zero) or omitted — Users can select and edit text, but all changes are tracked with revision marks. 1 — Users can only add annotations. 2 — Users can only select and modify text in form fields. To specify which sections of a multiple-section form should be protected and which should not, use **ToolsProtectSection**.

Unlike **FileSaveAs** or **ToolsOptionsSave**, which you can use to protect a document from being opened or saved, **ToolsProtectDocument** controls what the user is allowed to do while the document is open.

See also

Forms Statements and Functions

FileSaveAs

ToolsOptionsSave

ToolsProtectSection

ToolsUnprotectDocument

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsProtectSection Example

This example protects a document for forms, but disables protection for the second section of the document:

```
ToolsProtectDocument .Type = 2
ToolsProtectSection .Section = 2, .Protect = 0
```

ToolsProtectSection

¬Example

ToolsProtectSection .Protect = *number* [**, .Section =** *number*]

Enables or disables protection for sections within a document when the document is protected for forms with **ToolsProtectDocument**. **ToolsProtectSection** has no effect if the document is protected for annotations or revision marks.

Argument	Explanation
.Protect	Specifies whether to enable or disable protection for the specified section: 0 (zero) or omitted Disables protection. 1 Enables protection.
.Section	The section for which you want to enable or disable protection: 1 corresponds to the first section, 2 to the second section, and so on.

See also

Forms Statements and Functions

ToolsProtectDocument

ToolsUnprotectDocument

ToolsUnprotectDocument

ToolsUnprotectDocument [**.DocumentPassword =** *text*]

Removes protection from the active document. If the document is not protected, an error occurs.

Argument	Explanation
.DocumentPassword	The password, if any, used to protect the document. Note that passwords are case-sensitive. If the document is protected with a password and you don't specify .DocumentPassword, a dialog box prompts the user for the password.

See also

Forms Statements and Functions

ToolsProtectDocument

ToolsProtectSection

Help

Help

Displays Help for the selected context. If no context is active --- for example, if the insertion point is in a Word document and not in a field --- **Help** displays the Word Help Contents screen.

_____

See also

Help Statements and Functions

HelpActiveWindow

HelpTool

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

HelpAbout Example

This example defines a dialog record for the About Microsoft Word dialog box, stores the current values in the dialog record, and then displays in a message box the value for the serial number:

```
Dim dlg As HelpAbout
GetCurValues dlg
MsgBox "Serial number: " + dlg.AppSerialNumber
```

HelpAbout

¬Example

HelpAbout [**.AppName =** *text*] [**, .AppCopyright =** *text*] [**, .AppUserName =** *text*] [**, .AppOrganization =** *text*] [**, .AppSerialNumber =** *text*]

Without any arguments, displays the About Microsoft Word dialog box (Help menu), which gives the Word version number, the serial number, copyright information, and the name of the licensed user.

The arguments for the **HelpAbout** statement are read-only, which means that unlike other WordBasic statements, **HelpAbout** cannot be used to set a value. Instead, you define a dialog record as **HelpAbout** and then use the dialog record in the same way you use a function: to return information. In this case, the dialog record can return information from the About Microsoft Word dialog box.

Argument	Explanation
.AppName	The name and version number of Word as they appear in the About Microsoft Word dialog box
.AppCopyright	The copyright notice for Word
.AppUserName	The name of the person to whom the active copy of Word is licensed
.AppOrganization	The organization name provided when Word was installed
.AppSerialNumber	The serial number of the active copy of Word

See also

Help Statements and Functions

AppInfo$()

DocumentStatistics

GetSystemInfo

MicrosoftSystemInfo

HelpActiveWindow

HelpActiveWindow

Displays a Help topic describing the command associated with the active view or pane.

_____

See also

Help Statements and Functions

Help

HelpTool

HelpContents

HelpContents

Displays the Word Help Contents screen.

_____

See also

Help Statements and Functions

Help

HelpIndex

HelpExamplesAndDemos

HelpExamplesAndDemos

Displays a screen from which you can access all the examples and demonstrations in Help.

_____

See also

Help Statements and Functions

HelpQuickPreview

HelpIndex

HelpIndex

Displays the index for Help.

See also

Help Statements and Functions

Help

HelpContents

HelpKeyboard

HelpKeyboard

Displays a list of Help topics about keyboard and mouse shortcuts.

HelpPSSHelp

HelpPSSHelp

Displays information in Help about the support services available for Microsoft Word.

HelpQuickPreview

HelpQuickPreview

Starts a tutorial that introduces Microsoft Word.

See also

Help Statements and Functions

HelpExamplesAndDemos

HelpSearch

HelpSearch

Displays the Search dialog box for performing a keyword search in Help.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

HelpTipOfTheDay Example

This example displays the Tip Of The Day dialog box:

```
Dim dlg As HelpTipOfTheDay
x = Dialog(dlg)
```

HelpTipOfTheDay

¬Example

HelpTipOfTheDay .StartupTips = *number*

Controls whether the Tip Of The Day dialog box appears when you start Word. To displays the Tip Of The Day dialog box, you can use a **Dialog** or **Dialog()** instruction as shown in the example for this reference entry.

Argument	Explanation
.StartupTips	If 1, displays the Tip Of The Day dialog box each time you start Word.

HelpTool

HelpTool

Changes the mouse pointer to a question mark, indicating you will get context-sensitive Help on the next command you choose or the next element of the Word screen you click. If you click text, Word displays a box describing current paragraph and character formats.

_____

See also

Help Statements and Functions

Help

HelpActiveWindow

HelpUsingHelp

HelpUsingHelp

Displays a list of Help topics that describe how to use Help.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

HelpWordPerfectHelp Example

This example displays the Help For WordPerfect Users dialog box:

```
Dim dlg As HelpWordPerfectHelp
x = Dialog (dlg)
```

HelpWordPerfectHelp

¬Example

HelpWordPerfectHelp

Used with a **Dialog** or **Dialog()** instruction, displays the Help For WordPerfect Users dialog box (Help menu), as shown in the example for this entry.

_____

See also

Help Statements and Functions

HelpWordPerfectHelpOptions

HelpWordPerfectHelpOptions

HelpWordPerfectHelpOptions [**.CommandKeyHelp =** *number*] [**, .DocNavKeys =** *number*] [**, .MouseSimulation =** *number*] [**, .DemoGuidance =** *number*] [**, .DemoSpeed =** *number*] [**, .HelpType =** *number*]

Sets options in the Help for WordPerfect Users feature. The arguments for **HelpWordPerfectHelpOptions** correspond to the options in the Help Options dialog box (Help For WordPerfect Users command, Help menu).

Argument	Explanation
.CommandKeyHelp	If 1, Word evaluates keystrokes as WordPerfect keystrokes.
.DocNavKeys	If 1, the arrow keys and the PAGE UP, PAGE DOWN, HOME, END, and ESC keys function as they would in WordPerfect.
.MouseSimulation	If 1, the mouse pointer moves as Help for WordPerfect Users selects options during demonstrations.
.DemoGuidance	If 1, Help for WordPerfect Users displays help text when user input is required during demonstrations.
.DemoSpeed	Controls the speed of demonstrations: 0 (zero) Fast 1 Medium 2 Slow
.HelpType	Specifies whether Help for WordPerfect Users displays help text or demonstrates the WordPerfect command you choose: 0 (zero) Help text 1 Demonstration

See also

Help Statements and Functions

HelpWordPerfectHelp

Toolbar Button Images and Numbers

You can choose images for toolbar buttons by specifying the associated image number in AddButton and ChooseButtonImage instructions. The following topics show each image and its associated number.

Button Images 1-78

Button Images 79-156

Button Images 157-234

Button Images 235-312

Button Images 1-78

1	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:0:0:16:0}	27	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:0:0:16:0}	53	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:0:0:16:0}
2	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:16:0:16:0}	28	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:16:0:16:0}	54	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:16:0:16:0}
3	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:32:0:16:0}	29	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:32:0:16:0}	55	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:32:0:16:0}
4	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:48:0:16:0}	30	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:48:0:16:0}	56	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:48:0:16:0}
5	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:64:0:16:0}	31	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:64:0:16:0}	57	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:64:0:16:0}
6	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:80:0:16:0}	32	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:80:0:16:0}	58	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:80:0:16:0}
7	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:96:0:16:0}	33	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:96:0:16:0}	59	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:96:0:16:0}
8	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:112:0	34	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:112:0	60	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:112:0

	:16:0}		:16:0}		:16:0}
9	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:128:0 :16:0}	35	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:128:0 :16:0}	61	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:128:0 :16:0}
10	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:144:0 :16:0}	36	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:144:0 :16:0}	62	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:144:0 :16:0}
11	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:160:0 :16:0}	37	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:160:0 :16:0}	63	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:160:0 :16:0}
12	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:176:0 :16:0}	38	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:176:0 :16:0}	64	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:176:0 :16:0}
13	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:192:0 :16:0}	39	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:192:0 :16:0}	65	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:192:0 :16:0}
14	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:208:0 :16:0}	40	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:208:0 :16:0}	66	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:208:0 :16:0}
15	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:224:0 :16:0}	41	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:224:0 :16:0}	67	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:224:0 :16:0}
16	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:240:0 :16:0}	42	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:240:0 :16:0}	68	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:240:0 :16:0}
17	{ewc	43	{ewc	69	{ewc

	shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:256:0 :16:0}		shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:256:0 :16:0}		shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:256:0 :16:0}
18	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:272:0 :16:0}	44	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:272:0 :16:0}	70	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:272:0 :16:0}
19	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:288:0 :16:0}	45	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:288:0 :16:0}	71	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:288:0 :16:0}
20	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:304:0 :16:0}	46	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:304:0 :16:0}	72	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:304:0 :16:0}
21	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:320:0 :16:0}	47	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:320:0 :16:0}	73	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:320:0 :16:0}
22	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:336:0 :16:0}	48	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:336:0 :16:0}	74	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:336:0 :16:0}
23	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:352:0 :16:0}	49	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:352:0 :16:0}	75	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:352:0 :16:0}
24	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR1:368:0 :16:0}	50	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR2:368:0 :16:0}	76	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR3:368:0 :16:0}
25	{ewc shareres, T3EWCL ASS,	51	{ewc shareres, T3EWCL ASS,	77	{ewc shareres, T3EWCL ASS,

	dllres:wordres.dll:TBAR1:384:0:16:0}		dllres:wordres.dll:TBAR2:384:0:16:0}		dllres:wordres.dll:TBAR3:384:0:16:0}
26	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR1:400:0:16:0}	52	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR2:400:0:16:0}	78	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR3:400:0:16:0}

See also

Button Images 79-156

Button Images 157-234

Button Images 235-312

Button Images 79-156

79	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:0:0:16:0}	105	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:0:0:16:0}	131	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:0:0:16:0}
80	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:16:0:16:0}	106	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:16:0:16:0}	132	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:16:0:16:0}
81	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:32:0:16:0}	107	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:32:0:16:0}	133	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:32:0:16:0}
82	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:48:0:16:0}	108	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:48:0:16:0}	134	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:48:0:16:0}
83	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:64:0:16:0}	109	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:64:0:16:0}	135	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:64:0:16:0}
84	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:80:0:16:0}	110	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:80:0:16:0}	136	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:80:0:16:0}
85	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:96:0:16:0}	111	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:96:0:16:0}	137	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:96:0:16:0}
86	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:112:0	112	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:112:0	138	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:112:0

:16:0}

87 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:128:0:16:0}

88 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:144:0:16:0}

89 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:160:0:16:0}

90 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:176:0:16:0}

91 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:192:0:16:0}

92 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:208:0:16:0}

93 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:224:0:16:0}

94 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:240:0:16:0}

95 {ewc

:16:0}

113 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:128:0:16:0}

114 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:144:0:16:0}

115 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:160:0:16:0}

116 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:176:0:16:0}

117 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:192:0:16:0}

118 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:208:0:16:0}

119 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:224:0:16:0}

120 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:240:0:16:0}

121 {ewc

:16:0}

139 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:128:0:16:0}

140 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:144:0:16:0}

141 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:160:0:16:0}

142 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:176:0:16:0}

143 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:192:0:16:0}

144 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:208:0:16:0}

145 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:224:0:16:0}

146 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:240:0:16:0}

147 {ewc

shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:256:0 :16:0}

96 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:272:0 :16:0}

97 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:288:0 :16:0}

98 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:304:0 :16:0}

99 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:320:0 :16:0}

100 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:336:0 :16:0}

101 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:352:0 :16:0}

102 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR4:368:0 :16:0}

103 {ewc shareres, T3EWCL ASS,

shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:256:0 :16:0}

122 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:272:0 :16:0}

123 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:288:0 :16:0}

124 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:304:0 :16:0}

125 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:320:0 :16:0}

126 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:336:0 :16:0}

127 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:352:0 :16:0}

128 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR5:368:0 :16:0}

129 {ewc shareres, T3EWCL ASS,

shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:256:0 :16:0}

148 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:272:0 :16:0}

149 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:288:0 :16:0}

150 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:304:0 :16:0}

151 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:320:0 :16:0}

152 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:336:0 :16:0}

153 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:352:0 :16:0}

154 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR6:368:0 :16:0}

155 {ewc shareres, T3EWCL ASS,

	dllres:wordres.dll:TBAR4:384:0:16:0}		dllres:wordres.dll:TBAR5:384:0:16:0}		dllres:wordres.dll:TBAR6:384:0:16:0}
104	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR4:400:0:16:0}	130	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR5:400:0:16:0}	156	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR6:400:0:16:0}

See also

Button Images 1-78

Button Images 157-234

Button Images 235-312

Button Images 157-234

157 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:0:0:16:0}

158 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:16:0:16:0}

159 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:32:0:16:0}

160 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:48:0:16:0}

161 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:64:0:16:0}

162 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:80:0:16:0}

163 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:96:0:16:0}

164 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:112:0

183 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:0:0:16:0}

184 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:16:0:16:0}

185 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:32:0:16:0}

186 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:48:0:16:0}

187 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:64:0:16:0}

188 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:80:0:16:0}

189 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:96:0:16:0}

190 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:112:0

209 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:0:0:16:0}

210 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:16:0:16:0}

211 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:32:0:16:0}

212 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:48:0:16:0}

213 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:64:0:16:0}

214 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:80:0:16:0}

215 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:96:0:16:0}

216 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:112:0

:16:0}

165 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:128:0:16:0}

166 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:144:0:16:0}

167 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:160:0:16:0}

168 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:176:0:16:0}

169 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:192:0:16:0}

170 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:208:0:16:0}

171 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:224:0:16:0}

172 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:240:0:16:0}

173 {ewc

:16:0}

191 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:128:0:16:0}

192 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:144:0:16:0}

193 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:160:0:16:0}

194 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:176:0:16:0}

195 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:192:0:16:0}

196 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:208:0:16:0}

197 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:224:0:16:0}

198 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:240:0:16:0}

199 {ewc

:16:0}

217 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:128:0:16:0}

218 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:144:0:16:0}

219 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:160:0:16:0}

220 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:176:0:16:0}

221 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:192:0:16:0}

222 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:208:0:16:0}

223 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:224:0:16:0}

224 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:240:0:16:0}

225 {ewc

shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:256:0 :16:0}

174 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:272:0 :16:0}

175 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:288:0 :16:0}

176 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:304:0 :16:0}

177 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:320:0 :16:0}

178 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:336:0 :16:0}

179 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:352:0 :16:0}

180 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR7:368:0 :16:0}

181 {ewc shareres, T3EWCL ASS,

shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:256:0 :16:0}

200 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:272:0 :16:0}

201 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:288:0 :16:0}

202 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:304:0 :16:0}

203 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:320:0 :16:0}

204 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:336:0 :16:0}

205 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:352:0 :16:0}

206 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR8:368:0 :16:0}

207 {ewc shareres, T3EWCL ASS,

shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:256:0 :16:0}

226 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:272:0 :16:0}

227 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:288:0 :16:0}

228 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:304:0 :16:0}

229 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:320:0 :16:0}

230 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:336:0 :16:0}

231 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:352:0 :16:0}

232 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR9:368:0 :16:0}

233 {ewc shareres, T3EWCL ASS,

	dllres:wordres.dll:TBAR7:384:0:16:0}		dllres:wordres.dll:TBAR8:384:0:16:0}		dllres:wordres.dll:TBAR9:384:0:16:0}
182	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR7:400:0:16:0}	208	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR8:400:0:16:0}	234	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR9:400:0:16:0}

See also

Button Images 1-78

Button Images 79-156

Button Images 235-312

Button Images 235-312

235 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:0:0:16:0}

236 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:16:0:16:0}

237 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:32:0:16:0}

238 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:48:0:16:0}

239 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:64:0:16:0}

240 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:80:0:16:0}

241 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:96:0:16:0}

242 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:112:

261 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:0:0:16:0}

262 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:16:0:16:0}

263 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:32:0:16:0}

264 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:48:0:16:0}

265 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:64:0:16:0}

266 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:80:0:16:0}

267 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:96:0:16:0}

268 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:112:

287 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:0:0:16:0}

288 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:16:0:16:0}

289 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:32:0:16:0}

290 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:48:0:16:0}

291 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:64:0:16:0}

292 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:80:0:16:0}

293 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:96:0:16:0}

294 {ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:112:

0:16:0}

243 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:128: 0:16:0}

244 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:144: 0:16:0}

245 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:160: 0:16:0}

246 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:176: 0:16:0}

247 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:192: 0:16:0}

248 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:208: 0:16:0}

249 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:224: 0:16:0}

250 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:240: 0:16:0}

251 {ewc

0:16:0}

269 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:128: 0:16:0}

270 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:144: 0:16:0}

271 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:160: 0:16:0}

272 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:176: 0:16:0}

273 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:192: 0:16:0}

274 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:208: 0:16:0}

275 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:224: 0:16:0}

276 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:240: 0:16:0}

277 {ewc

0:16:0}

295 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:128: 0:16:0}

296 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:144: 0:16:0}

297 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:160: 0:16:0}

298 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:176: 0:16:0}

299 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:192: 0:16:0}

300 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:208: 0:16:0}

301 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:224: 0:16:0}

302 {ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:240: 0:16:0}

303 {ewc

	shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:256: 0:16:0}		shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:256: 0:16:0}		shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:256: 0:16:0}
252	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:272: 0:16:0}	278	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:272: 0:16:0}	304	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:272: 0:16:0}
253	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:288: 0:16:0}	279	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:288: 0:16:0}	305	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:288: 0:16:0}
254	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:304: 0:16:0}	280	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:304: 0:16:0}	306	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:304: 0:16:0}
255	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:320: 0:16:0}	281	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:320: 0:16:0}	307	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:320: 0:16:0}
256	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:336: 0:16:0}	282	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:336: 0:16:0}	308	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:336: 0:16:0}
257	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:352: 0:16:0}	283	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:352: 0:16:0}	309	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:352: 0:16:0}
258	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR10:368: 0:16:0}	284	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR11:368: 0:16:0}	310	{ewc shareres, T3EWCL ASS, dllres:word res.dll:TB AR12:368: 0:16:0}
259	{ewc shareres, T3EWCL ASS,	285	{ewc shareres, T3EWCL ASS,	311	{ewc shareres, T3EWCL ASS,

	dllres:wordres.dll:TBAR10:384:0:16:0}		dllres:wordres.dll:TBAR11:384:0:16:0}		dllres:wordres.dll:TBAR12:384:0:16:0}
260	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR10:400:0:16:0}	286	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR11:400:0:16:0}	312	{ewc shareres, T3EWCLASS, dllres:wordres.dll:TBAR12:400:0:16:0}

See also

Button Images 1-78

Button Images 79-156

Button Images 157-234

WordBasic Statements and Functions Index

-A-

Abs()
Activate
ActivateObject
AddAddIn, AddAddIn()
AddButton
AddDropDownItem
AddInState, AddInState()
AllCaps, AllCaps()
AnnotationRefFromSel$()
AppActivate
AppClose
AppCount()
AppGetNames, AppGetNames()
AppHide
AppInfo$()
AppIsRunning()
AppMaximize, AppMaximize()

AppMinimize, AppMinimize()
AppMove
AppRestore, AppRestore()
AppSendMessage
AppShow
AppSize
AppWindowHeight, AppWindowHeight()
AppWindowPosLeft, AppWindowPosLeft()
AppWindowPosTop, AppWindowPosTop()
AppWindowWidth, AppWindowWidth()
Asc()
AtEndOfDocument()
AtStartOfDocument()
AutoMarkIndexEntries
AutoText
AutoTextName$()

-B-

Beep
Begin Dialog...End Dialog
Bold, Bold()
BookmarkName$()
BorderBottom, BorderBottom()
BorderInside, BorderInside()
BorderLeft, BorderLeft()
BorderLineStyle, BorderLineStyle()
BorderNone, BorderNone()
BorderOutside, BorderOutside()
BorderRight, BorderRight()
BorderTop, BorderTop()

-C-

Call
Cancel
CancelButton
CenterPara, CenterPara()
ChangeCase, ChangeCase()
CharColor, CharColor()
CharLeft, CharLeft()
CharRight, CharRight()
ChDefaultDir
ChDir

CheckBox
CheckBoxFormField
ChooseButtonImage
Chr$()
CleanString$()
ClearAddIns
ClearFormField
Close
ClosePane
ClosePreview
CloseUpPara
CloseViewHeaderFooter
CmpBookmarks()
ColumnSelect
ComboBox
CommandValid()
Connect
ControlRun
Converter$()
ConverterLookup()
ConvertObject
CopyBookmark
CopyButtonImage
CopyFile
CopyFormat
CopyText
CountAddIns()
CountAutoTextEntries()
CountBookmarks()
CountDirectories()
CountDocumentVars()
CountFiles()
CountFonts()
CountFoundFiles()
CountKeys()
CountLanguages()
CountMacros()
CountMenuItems()
CountMenus()
CountMergeFields()
CountStyles()
CountToolbarButtons()

CountToolbars()
CountToolsGrammarStatistics()
CountWindows()
CreateSubdocument

-D-

Date$()
DateSerial()
DateValue()
Day()
Days360()
DDEExecute
DDEInitiate()
DDEPoke
DDERequest$()
DDETerminate
DDETerminateAll
Declare
DefaultDir$()
DeleteAddIn
DeleteBackWord
DeleteButton
DeleteWord
DemoteList
DemoteToBodyText
Dialog, Dialog()
DialogEditor
Dim
DisableAutoMacros
DisableInput
DlgControlId()
DlgEnable, DlgEnable()
DlgFilePreview, DlgFilePreview$()
DlgFocus, DlgFocus$()
DlgListBoxArray, DlgListBoxArray()
DlgSetPicture
DlgText, DlgText$()
DlgUpdateFilePreview
DlgValue, DlgValue()
DlgVisible, DlgVisible()
DocClose
DocMaximize, DocMaximize()

DocMinimize, DocMinimize()
DocMove
DocRestore
DocSize
DocSplit, DocSplit()
DocumentStatistics
DocWindowHeight, DocWindowHeight()
DocWindowPosLeft, DocWindowPosLeft()
DocWindowPosTop, DocWindowPosTop()
DocWindowWidth, DocWindowWidth()
DoFieldClick
DOSToWin$()
DottedUnderline, DottedUnderline()
DoubleUnderline, DoubleUnderline()
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
DropDownFormField
DropListBox

-E-

EditAutoText
EditBookmark
EditButtonImage
EditClear
EditConvertAllEndnotes
EditConvertAllFootnotes
EditConvertNotes
EditCopy
EditCut
EditFind
EditFindClearFormatting
EditFindFont
EditFindFound()
EditFindLang
EditFindPara
EditFindStyle

EditGoTo
EditLinks
EditObject
EditPaste
EditPasteSpecial
EditPicture
EditRedo
EditRepeat
EditReplace
EditReplaceClearFormatting
EditReplaceFont
EditReplaceLang
EditReplacePara
EditReplaceStyle
EditSelectAll
EditSwapAllNotes
EditTOACategory
EditUndo
EmptyBookmark()
EnableFormField
EndOfColumn, EndofColumn()
EndOfDocument, EndOfDocument()
EndOfLine, EndOfLine()
EndOfRow, EndOfRow()
EndOfWindow, EndOfWindow()
Environ$()
Eof()
Err
Error
ExistingBookmark()
ExitWindows
ExtendMode()
ExtendSelection

-F-

FieldSeparator, FieldSeparator$()
FileClose
FileCloseAll
FileClosePicture
FileConfirmConversions, FileConfirmConversions()
FileExit
FileFind

FileList
FileName$()
FileNameFromWindow$()
FileNameInfo$()
FileNew
FileNewDefault
File*Number*
FileOpen
FilePageSetup
FilePreview
FilePrint
FilePrintDefault
FilePrintPreview, FilePrintPreview()
FilePrintPreviewFullScreen
FilePrintPreviewPages, FilePrintPreviewPages()
FilePrintSetup
FileRoutingSlip
Files$()
FileSave
FileSaveAll
FileSaveAs
FileSendMail
FileSummaryInfo
FileTemplates
Font, Font$()
FontSize, FontSize()
FontSizeSelect
FontSubstitution
For...Next
FormatAddrFonts
FormatAutoFormat
FormatBordersAndShading
FormatBullet
FormatBulletDefault, FormatBulletDefault()
FormatBulletsAndNumbering
FormatCallout
FormatChangeCase
FormatColumns
FormatDefineStyleBorders
FormatDefineStyleFont
FormatDefineStyleFrame
FormatDefineStyleLang

FormatDefineStyleNumbers
FormatDefineStylePara
FormatDefineStyleTabs
FormatDrawingObject
FormatDropCap
FormatFont
FormatFrame
FormatHeaderFooterLink
FormatHeadingNumber
FormatHeadingNumbering
FormatMultilevel
FormatNumber
FormatNumberDefault, FormatNumberDefault()
FormatPageNumber
FormatParagraph
FormatPicture
FormatRetAddrFonts
FormatSectionLayout
FormatStyle
FormatStyleGallery
FormatTabs
FormFieldOptions
FormShading, FormShading()
FoundFileName$()
Function...End Function

-G-

GetAddInID()
GetAddInName$()
GetAttr()
GetAutoCorrect$()
GetAutoText$()
GetBookmark$()
GetCurValues
GetDirectory$()
GetDocumentVar$()
GetDocumentVarName$()
GetFieldData$()
GetFormResult(), GetFormResult$()
GetMergeField$()
GetPrivateProfileString$()
GetProfileString$()

GetSelEndPos()
GetSelStartPos()
GetSystemInfo, GetSystemInfo$()
GetText$()
GoBack
Goto
GoToAnnotationScope
GoToHeaderFooter
GoToNext*Item*
GoToPrevious*Item*
GroupBox
GrowFont
GrowFontOnePoint

-H-

HangingIndent
Help
HelpAbout
HelpActiveWindow
HelpContents
HelpExamplesAndDemos
HelpIndex
HelpKeyboard
HelpPSSHelp
HelpQuickPreview
HelpSearch
HelpTipOfTheDay
HelpTool
HelpUsingHelp
HelpWordPerfectHelp
HelpWordPerfectHelpOptions
Hidden, Hidden()
HLine
Hour()
HPage
HScroll, HScroll()

-I-

If...Then...Else
Indent
Input
Input$()

InputBox$()
Insert
InsertAddCaption
InsertAnnotation
InsertAutoCaption
InsertAutoText
InsertBreak
InsertCaption
InsertCaptionNumbering
InsertChart
InsertColumnBreak
InsertCrossReference
InsertDatabase
InsertDateField
InsertDateTime
InsertDrawing
InsertEquation
InsertExcelTable
InsertField
InsertFieldChars
InsertFile
InsertFootnote
InsertFormField
InsertFrame
InsertIndex
InsertMergeField
InsertObject
InsertPageBreak
InsertPageField
InsertPageNumbers
InsertPara
InsertPicture
InsertSectionBreak
InsertSound
InsertSpike
InsertSubdocument
InsertSymbol
InsertTableOfAuthorities
InsertTableOfContents
InsertTableOfFigures
InsertTimeField
InsertWordArt

InStr()
Int()
IsDocumentDirty()
IsExecuteOnly()
IsMacro()
IsTemplateDirty()
Italic, Italic()

-J-

JustifyPara, JustifyPara()

-K-

KeyCode()
KeyMacro$()
Kill

-L-

Language, Language$()
LCase$()
Left$()
LeftPara, LeftPara()
Len()
Let
Line Input
LineDown, LineDown()
LineUp, LineUp()
ListBox
LockDocument, LockDocument()
LockFields
Lof()
LTrim$()

-M-

MacroCopy
MacroDesc$()
MacroFileName$()
MacroName$()
MacroNameFromWindow$()
Magnifier, Magnifier()
MailMerge
MailMergeAskToConvertChevrons, MailMergeAskToConvertChevrons()
MailMergeCheck

MailMergeConvertChevrons, MailMergeConvertChevrons()
MailMergeCreateDataSource
MailMergeCreateHeaderSource
MailMergeDataForm
MailMergeDataSource$()
MailMergeEditDataSource
MailMergeEditHeaderSource
MailMergeEditMainDocument
MailMergeFindRecord
MailMergeFirstRecord
MailMergeFoundRecord()
MailMergeGotoRecord, MailMergeGotoRecord()
MailMergeHelper
MailMergeInsertAsk
MailMergeInsertFillIn
MailMergeInsertIf
MailMergeInsertMergeRec
MailMergeInsertMergeSeq
MailMergeInsertNext
MailMergeInsertNextIf
MailMergeInsertSet
MailMergeInsertSkipIf
MailMergeLastRecord
MailMergeMainDocumentType, MailMergeMainDocumentType()
MailMergeNextRecord
MailMergeOpenDataSource
MailMergeOpenHeaderSource
MailMergePrevRecord
MailMergeQueryOptions
MailMergeReset
MailMergeState()
MailMergeToDoc
MailMergeToPrinter
MailMergeViewData, MailMergeViewData()
MarkCitation
MarkIndexEntry
MarkTableOfContentsEntry
MenuItemMacro$()
MenuItemText$()
MenuMode
MenuText$()
MergeFieldName$()

MergeSubdocument
MicrosoftAccess
MicrosoftExcel
MicrosoftFoxPro
MicrosoftMail
MicrosoftPowerPoint
MicrosoftProject
MicrosoftPublisher
MicrosoftSchedule
MicrosoftSystemInfo
Mid$()
Minute()
MkDir
Month()
MoveButton
MoveText
MoveToolbar
MsgBox, MsgBox()

-N-

Name
NewToolbar
NextCell, NextCell()
NextField, NextField()
NextObject
NextPage, NextPage()
NextTab()
NextWindow
NormalFontPosition
NormalFontSpacing
NormalStyle
NormalViewHeaderArea
NoteOptions
Now()

-O-

OK
OKButton
On Error
OnTime
Open
OpenSubdocument

OpenUpPara
OptionButton
OptionGroup
Organizer
OtherPane
OutlineCollapse
OutlineDemote
OutlineExpand
OutlineLevel()
OutlineMoveDown
OutlineMoveUp
OutlinePromote
OutlineShowFirstLine, OutlineShowFirstLine()
OutlineShowFormat
Overtype, Overtype()

-P-

PageDown, PageDown()
PageUp, PageUp()
ParaDown, ParaDown()
ParaKeepLinesTogether, ParaKeepLinesTogether()
ParaKeepWithNext, ParaKeepWithNext()
ParaPageBreakBefore, ParaPageBreakBefore()
ParaUp, ParaUp()
ParaWidowOrphanControl, ParaWidowOrphanControl()
PasteButtonImage
PasteFormat
PathFromMacPath$()
PauseRecorder
Picture
PrevCell, PrevCell()
PrevField, PrevField()
PrevObject
PrevPage, PrevPage()
PrevTab()
PrevWindow
Print
PromoteList
PushButton
PutFieldData

-R-

Read
Redim
REM
RemoveAllDropDownItems
RemoveBulletsNumbers
RemoveDropDownItem
RemoveFrames
RemoveSubdocument
RenameMenu
RepeatFind
ResetButtonImage
ResetChar, ResetChar()
ResetNoteSepOrNotice
ResetPara, ResetPara()
Right$()
RightPara, RightPara()
RmDir
Rnd()
RTrim$()
RunPrintManager

-S-

SaveTemplate
ScreenRefresh
ScreenUpdating, ScreenUpdating()
Second()
Seek, Seek()
Select Case
SelectCurAlignment
SelectCurColor
SelectCurFont
SelectCurIndent
SelectCurSentence
SelectCurSpacing
SelectCurTabs
SelectCurWord
SelectDrawingObjects
Selection$()
SelectionFileName$()
SelInfo()
SelType, SelType()
SendKeys

SentLeft, SentLeft()
SentRight, SentRight()
SetAttr
SetAutoText
SetDocumentDirty
SetDocumentVar, SetDocumentVar()
SetEndOfBookmark
SetFormResult
SetPrivateProfileString, SetPrivateProfileString()
SetProfileString, SetProfileString()
SetSelRange
SetStartOfBookmark
SetTemplateDirty
Sgn()
ShadingPattern, ShadingPattern()
Shell
ShowAll, ShowAll()
ShowAllHeadings
ShowAnnotationBy
ShowHeading*Number*
ShowNextHeaderFooter
ShowPrevHeaderFooter
ShowVars
ShrinkFont
ShrinkFontOnePoint
ShrinkSelection
SizeToolbar
SkipNumbering, SkipNumbering()
SmallCaps, SmallCaps()
SortArray
SpacePara1, SpacePara1()
SpacePara15, SpacePara15()
SpacePara2, SpacePara2()
Spike
SplitSubdocument
StartOfColumn, StartOfColumn()
StartOfDocument, StartOfDocument()
StartOfLine, StartOfLine()
StartOfRow, StartOfRow()
StartOfWindow, StartOfWindow()
Stop
Str$()

Strikethrough, Strikethrough()
String$()
Style
StyleDesc$()
StyleName$()
Sub...End Sub
Subscript, Subscript()
Superscript, Superscript()
SymbolFont

-T-

TabLeader$()
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
TabType()
Text
TextBox
TextFormField
TextToTable

Time$()
TimeSerial()
TimeValue()
Today()
ToggleFieldDisplay
ToggleFull
ToggleHeaderFooterLink
ToggleMainTextLayer
TogglePortrait
ToggleScribbleMode
ToolbarButtonMacro$()
ToolbarName$()
ToolbarState()
ToolsAddRecordDefault
ToolsAdvancedSettings
ToolsAutoCorrect
ToolsAutoCorrectDays, ToolsAutoCorrectDays()
ToolsAutoCorrectInitialCaps, ToolsAutoCorrectInitialCaps()
ToolsAutoCorrectReplaceText, ToolsAutoCorrectReplaceText()
ToolsAutoCorrectSentenceCaps, ToolsAutoCorrectSentenceCaps()
ToolsAutoCorrectSmartQuotes, ToolsAutoCorrectSmartQuotes()
ToolsBulletListDefault
ToolsBulletsNumbers
ToolsCalculate, ToolsCalculate()
ToolsCompareVersions
ToolsCreateEnvelope
ToolsCreateLabels
ToolsCustomize
ToolsCustomizeKeyboard
ToolsCustomizeMenuBar
ToolsCustomizeMenus
ToolsGetSpelling, ToolsGetSpelling()
ToolsGetSynonyms, ToolsGetSynonyms()
ToolsGrammar
ToolsGrammarStatisticsArray
ToolsHyphenation
ToolsHyphenationManual
ToolsLanguage
ToolsMacro
ToolsManageFields
ToolsMergeRevisions
ToolsNumberListDefault

ToolsOptions
ToolsOptionsAutoFormat
ToolsOptionsCompatibility
ToolsOptionsEdit
ToolsOptionsFileLocations
ToolsOptionsGeneral
ToolsOptionsGrammar
ToolsOptionsPrint
ToolsOptionsRevisions
ToolsOptionsSave
ToolsOptionsSpelling
ToolsOptionsUserInfo
ToolsOptionsView
ToolsProtectDocument
ToolsProtectSection
ToolsRemoveRecordDefault
ToolsRepaginate
ToolsReviewRevisions
ToolsRevisionAuthor$()
ToolsRevisionDate()
ToolsRevisionDate$()
ToolsRevisions
ToolsRevisionType()
ToolsShrinkToFit
ToolsSpelling
ToolsSpellSelection
ToolsThesaurus
ToolsUnprotectDocument
ToolsWordCount

-U-

UCase$()
Underline, Underline()
UnHang
UnIndent
UnlinkFields
UnlockFields
UpdateFields
UpdateSource

-V-

Val()

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
VLine
VPage
VScroll, VScroll()

-W-

WaitCursor
Weekday()
While...Wend
Window()
WindowArrangeAll

WindowList
WindowName$()
WindowNewWindow
Window*Number*
WindowPane()
WinToDOS$()
WordLeft, WordLeft()
WordRight, WordRight()
WordUnderline, WordUnderline()
Write

-Y-

Year()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CommandValid() Example

This example tests whether the Undo command (Edit menu) is available. If it is, the previous editing or formatting command is undone.

```
If CommandValid("EditUndo") Then
    EditUndo
Else
    MsgBox "Command not available."
End If
```

CommandValid()

¬Example

CommandValid(*CommandName$***)**

Returns -1 if *CommandName$* is available in the current context; returns 0 (zero) if the command is not available or if *CommandName$* is not a WordBasic statement that corresponds to a Word command or dialog box option.

_____

See also

Macro Statements and Functions

IsMacro()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountMacros() Example

This example stores the number of built-in commands in builtins and then inserts in the active document a list of the names of the built-in commands:

```
loaded = CountMacros(0, 0, 1)
active = CountMacros(1)
normal = CountMacros(0)
templates = active + normal
nonbuiltins = loaded + templates
builtins = CountMacros(0, 1) - nonbuiltins
For count = 1 to builtins
    pos = count + nonbuiltins
    Insert Str$(count) + Chr$(9) + MacroName$(pos, 0, 1)
    InsertPara
Next count
```

CountMacros()

¬Example

CountMacros([*Context*] [**,** *All*] [**,** *Global*]**)**

Returns the number of macros available in the specified context.

Argument	Explanation
Context	Specifies the template in which to count macros: 0 (zero) or omitted Normal template 1 Active template Note that if you specify 1 and Normal is the active template, **CountMacros()** returns 0 (zero).
All	If 1, all available macros, add-in commands, and built-in commands are included in the count.
Global	If 1, only macros stored in loaded global templates and add-in commands are counted.

See also

Macro Statements and Functions

MacroName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DisableAutoMacros Example

This example creates a new document based on the Memo template. Assume that the Memo template contains an AutoNew macro that presents a series of dialog boxes. The **DisableAutoMacros** statement prevents AutoNew from running.

```
DisableAutoMacros
FileNew .Template = "MEMO"
```

DisableAutoMacros

¬Example

DisableAutoMacros [*Disable*]

Disables AutoOpen, AutoClose, AutoNew, and AutoExit macros until they are re-enabled with the instruction DisableAutoMacros 0 or until Word is started again. You can place a **DisableAutoMacros** statement before the line in your macro that triggers an auto macro --- for example, a **FileNew** instruction that creates a document based on a template containing an AutoNew macro.

Argument	Explanation
Disable	Specifies whether to disable auto macros: 0 (zero) Enables auto macros. 1 or omitted Disables auto macros.

You cannot use **DisableAutoMacros** to disable an AutoExec macro or to disable the current macro. For example, you could not place a **DisableAutoMacros** statement in an AutoOpen macro to disable AutoOpen. To disable an AutoExec macro, hold down the SHIFT key while starting Word in Program Manager, or use the following command line:

winword.exe /m

You can use the following command line to disable all auto macros, including AutoExec:

winword.exe /mDisableAutoMacros

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

IsMacro() Example

This example creates a new document if the active window is a macro-editing window:

```
If IsMacro() = -1 Then FileNewDefault
```

IsMacro()

¬Example

IsMacro([*WindowNumber*]**)**

Returns -1 if the window specified by *WindowNumber* is a macro-editing window and 0 (zero) if it is not.

Argument	Explanation
WindowNumber	Specifies a window on the Window menu: 1 corresponds to the first window, 2 to the second window, and so on. If *WindowNumber* is 0 (zero) or omitted, the active window is assumed. If *WindowNumber* does not correspond to a number on the Window menu, an error occurs.

See also

Macro Statements and Functions

MacroFileName$()

MacroNameFromWindow$()

SelInfo()

KeyMacro$()

KeyMacro$(*Count* [**,** *Context*]**)**

Returns the name of a macro or built-in command with a key assignment that differs from the default assignment.

Argument	Explanation
Count	A number in the range 1 to **CountKeys()** specifying the custom key assignment for which you want to return the command or macro name
Context	The template containing the custom key assignment: 0 (zero) or omitted Normal template 1 Active template

For an example, see KeyCode() Example.

See also

Macro Statements and Functions

CountKeys()

KeyCode()

MenuItemMacro$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MacroCopy Example

This example copies the MyTest macro from MYTEMP.DOT to the Normal template. The new macro is given the name MyFinal.

```
MacroCopy "C:\TEMPLATE\MYTEMP:MyTest", "Global:MyFinal"
```

MacroCopy

¬Example

MacroCopy [*Template1***:**]*Macro1$***,** [*Template2***:**]*Macro2$* [**,** *ExecuteOnly*]

Copies a macro from one open template to another. **MacroCopy** cannot replace an open macro. A template is open if it is attached to an open document, open in a document window, or loaded as a global template. If you make the new macro execute-only, it cannot be edited.

Argument	Explanation
Template1, Template2	The open template containing the macro you want to copy and the open template to which you want to copy the macro, respectively. If you do not include the extension in the template name, .DOT is assumed. Include paths if the templates are not in the directory specified by User Templates in the File Types box on the File Locations tab in the Options dialog box (Tools menu).
Macro1$, Macro2$	The name of the macro to copy and the name of the new macro, respectively.
ExecuteOnly	If nonzero, makes the destination macro execute-only. Note that this action cannot be reversed. It is a good idea to make a copy of a macro before making the original execute-only.

If you don't specify a source or destination template, the Normal template is assumed. You can use the syntax **Global:***MacroName$* to specify the Normal template as the source or destination in a **MacroCopy** instruction. For example:

```
MacroCopy "MyTemp:DelStylesTest", "Global:DelStyles"
```

Though not required, including "Global:" can make **MacroCopy** instructions affecting the Normal template easier to read.

See also

Macro Statements and Functions

IsExecuteOnly()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MacroDesc$() Example

This example prints a list of macros in the Normal template and their descriptions. To print a similar list for the active template, substitute CountMacros(1) and MacroName$(count, 1) in the first and second instructions, respectively.

```
For count = 1 To CountMacros(0)
    name$ = MacroName$(count, 0)
    Bold 1
    Insert name$ + Chr$(9)
    Bold 0
    Insert MacroDesc$(name$)
    InsertPara
Next count
```

MacroDesc$()

¬Example

MacroDesc$(***Name$*)**

Returns the description associated with the macro *Name$*. The same text is displayed in the Description box when you select the macro in the Macro dialog box (Tools menu) and in the status bar when you select a macro assigned to a menu or toolbar. You can set macro descriptions with **ToolsMacro**.

If *Name$* does not exist in the active template or the Normal template, an error occurs. If *Name$* exists in both the active template and the Normal template, the description for the macro in the active template is returned.

_____

See also

Macro Statements and Functions

CountMacros()

KeyMacro$()

MacroName$()

MenuItemMacro$()

ToolsMacro

MacroFileName$()

MacroFileName$(**[*MacroName$*])**

Returns the path and filename of the template containing the macro specified by *MacroName$*. Word first looks for the macro in the active template (if different from Normal), then in the Normal template, then in any loaded global templates (in alphabetic order), and finally in built-in commands (the same order Word would look for the macro when running it). If *MacroName$* is omitted, **MacroFileName$()** returns the path and filename of the template containing the macro that is running. If *MacroName$* does not exist, **MacroFileName$()** returns an empty string ("").

See also

Macro Statements and Functions

MacroDesc$()

MacroName$()

MacroNameFromWindow$()

MacroName$()

MacroName$(*Count* [**,** *Context*] [**,** *All*] [**,** *Global*]**)**

Returns the name of the macro defined in the specified context.

Argument	Explanation
Count	A number representing a position in the internal list of macros in the given context. Unlike lists in the Macro dialog box (Tools menu), the order is not alphabetic. The number can range from 1 to the number returned by **CountMacros()** in the given context. If you specify 0 (zero) for *Count*, **MacroName$()** returns the name of the macro in the active (or most recently active) macro-editing window.
Context	Specifies the template from which the internal list of macros is generated: 0 (zero) Normal template 1 Active template Note that if you specify 1 when Normal is the active template and *Count* is greater than 0, an error occurs.
All	If 1, all available macros, add-in commands, and built-in commands are listed in the following order: macros stored in the active template, macros stored in the Normal template, macros stored in loaded global templates, add-in commands, and built-in commands.
Global	If 1, only macros stored in loaded global templates and add-in commands are listed.

For an example that uses **MacroName$()**, see MacroDesc$() Example.

See also

Macro Statements and Functions

CountMacros()

MacroNameFromWindow$()

MacroNameFromWindow$([*WindowNumber*]**)**

Returns the name of the macro in the specified macro-editing window. If the specified window is not a macro-editing window, **MacroNameFromWindow$()** returns an empty string ("").

Argument	Explanation
WindowNumber	Specifies a window on the Window menu: 1 corresponds to the first window, 2 to the second window, and so on. If *WindowNumber* is 0 (zero) or omitted, the active window is assumed. If *WindowNumber* does not correspond to a number on the Window menu, an error occurs.

See also

Macro Statements and Functions

IsMacro()

MacroFileName$()

MacroName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MenuItemMacro$() Example

This example defines an array variable containing the names of commands and macros assigned to the File menu in the current working environment. To define a similar array for the File menu of the active template, substitute CountMenuItems("File", 0, 1) in the first instruction and MenuItemMacro$("File", 0, count, 1) in the fourth instruction.

```
numItems = CountMenuItems("File", 0, 0)
Dim fileItem$(numItems - 1)
For count = 1 To numItems
    fileItem$(count - 1) = MenuItemMacro$("File", 0, count, 0)
Next count
```

MenuItemMacro$()

¬Example

MenuItemMacro$(***Menu$*,** *Type***,** *Item* [**,** *Context*]**)**

Returns the name of the macro or built-in command associated with the specified menu item.

Argument	Explanation
Menu$	The name of a menu or shortcut menu. Menu names are listed in the Change What Menu box on the Menus tab in the Customize dialog box (Tools menu). Including an ampersand (&) before the underlined letter in the menu name is optional (for example, you can specify either "File" or "&File"). Do not include the parenthetical phrases "(No Document)" and "(Shortcut)", even though that text appears in the Customize dialog box.
Type	The type of menu: 0 Menus on the menu bar when a document is open 1 Menus on the menu bar when no document is open 2 Shortcut menus
Item	A number representing the item's position on the menu. The number can range from 1 to the number returned by **CountMenuItems()**. Separators between commands are considered items, and if *Item* represents the position of a separator, **MenuItemMacro$()** returns the string "(Separator)." Note that lists such as the list of recently used files on the File menu or proofing tools on the Tools menu correspond to a single item. If you specify the position of a list, **MenuItemMacro$()** returns an empty string ("").
Context	Specifies the menu assignment for which you want to return the macro or command name: 0 (zero) or omitted The assignment that is available when a document based on the Normal template is active 1 The assignment that is currently available Note that the assignment that is available depends on the custom settings, if any, of the active

template, any loaded global templates, and the Normal template.

See also

Macro Statements and Functions

CountMacros()

KeyMacro$()

MacroDesc$()

MacroName$()

MenuItemText$()

ToolsCustomizeMenus

PauseRecorder

PauseRecorder

Stops and restarts macro recording. **PauseRecorder** is the built-in Word command that runs when the user chooses the Pause Recorder button on the Macro toolbar; this statement would not typically appear in a macro.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

REM Example

This example demonstrates four ways to include a comment in a macro.

```
REM MyMacro Title -- MyName Here
'More about MyMacro
ParaDown        'Move to start of next paragraph
ParaUp          REM Moves to start of previous paragraph
```

REM

¬Example

REM *Comments*

'*Comments*

Designates explanatory text, which Word ignores when the macro is run. You can use an apostrophe (') instead of **REM**. Each notation has its advantages and disadvantages: **REM** makes it easier to distinguish comments from WordBasic instructions, but requires more space than an apostrophe.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ShowVars Example

This example defines the variables name$ and age as the items in the first line of the text file DATA.TXT. The variables are displayed in a dialog box where the values may be modified.

```
Open "C:\TMP\DATA.TXT" For Input As #1
Input #1, Name$, Age
ShowVars
```

ShowVars

¬Example

ShowVars

Displays a list of variables and their current values to help you debug the active macro. With **ShowVars** instructions, you can pause the macro and check the variables at exactly the points you want.

_____

See also

Macro Statements and Functions

MsgBox()

Print

Stop

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsMacro Example

This example opens a macro-editing window for the macro Test stored in the Normal template:

```
ToolsMacro .Name = "Test", .Show = 1, .Edit
```

ToolsMacro

¬Example

ToolsMacro .Name = *text* [**, .Run**] [**, .Edit**] [**, .Show =** *number*] [**, .Delete**] [**, .Rename**] [**, .Description =** *text*] [**, .NewName =** *text*] [**, .SetDesc**]

Runs or records a macro, sets the description of a macro, renames a macro, or opens a new or existing macro in a macro-editing window. The arguments for the **ToolsMacro** statement correspond to the options in the Macro dialog box (Tools menu). Note that the .Rename and .NewName arguments are included for compatibility with previous versions of Word.

Argument	Explanation
.Name	The name of the macro.
.Run	Runs the macro.
.Edit	Opens a macro-editing window containing the macro specified by .Name.
.Show	Specifies the context: 0 (zero) All available macros; includes macros in loaded global templates 1 Macros in the Normal template 2 Built-in commands 3 If the active document is not based on the Normal template, macros in the active template; otherwise, macros in the first loaded global template (global templates are listed in alphabetic order) > 3 Macros in the remaining loaded global templates (in alphabetic order) Omitted Word looks for the macro in whatever context was last selected in the Macro dialog box. To ensure that the correct macro runs, especially if a macro naming conflict between templates is possible, you should specify a value for .Show.
.Delete	Deletes the specified macro.
.Rename	Renames the specified macro.
.Description	Specifies a new description for the macro; used with .SetDesc. If the macro is assigned to a menu or toolbar and is selected or pointed to, the description is displayed in the status bar.
.NewName	A new name for the macro; used with .Rename.
.SetDesc	Sets a new description for the macro.

See also

Macro Statements and Functions

CountMacros()

IsMacro()
KeyMacro$()
MacroDesc$()
MacroFileName$()
MacroNameFromWindow$()
MenuItemMacro$()
Organizer
PauseRecorder

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountMergeFields() Example

This example inserts a list of merge field names in the active document:

```
For n = 1 To CountMergeFields()
    Insert MergeFieldName$(n)
    InsertPara
Next
```

CountMergeFields()

¬Example

CountMergeFields()

Returns the number of fields in the header record of the data source or in the header source attached to the active main document. **CountMergeFields()** returns 0 (zero) if the active document is not a main document, data source, or header source.

See also

Mail Merge Statements and Functions

InsertMergeField

MergeFieldName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetMergeField$() Example

This example displays the contents of the FirstName field for the current data record:

```
MsgBox GetMergeField$("FirstName")
```

GetMergeField$()

¬Example

GetMergeField$(***FieldName$*)**

Returns the contents of the specified merge field in the current data record. If the main document is not the active document or *FieldName$* doesn't exist, **GetMergeField$()** returns an empty string ("").

_____

See also

Mail Merge Statements and Functions

CountMergeFields()

MergeFieldName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertMergeField Example

This example inserts a MERGEFIELD field that is updated with the city name from the data source during a mail merge:

```
InsertMergeField .MergeField = "City"
```

InsertMergeField

¬Example

InsertMergeField .MergeField = *text*

Inserts a MERGEFIELD field that Word updates in each merge document during a mail merge. If the active document is not a main document, an error occurs.

Argument	Explanation
.MergeField	A merge field name corresponding to the type of information you want to insert. Field names appear in the first record of the data source or in the header source (if a header source is attached to the main document).

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertAsk

MailMergeInsertFillIn

MailMergeInsertIf

MailMergeInsertMergeRec

MailMergeInsertMergeSeq

MailMergeInsertNext

MailMergeInsertNextIf

MailMergeInsertSet

MailMergeInsertSkipIf

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MailMerge Example

This example merges the main document with data records 50 to 100 and sends the merge documents to a new document:

```
MailMerge .CheckErrors = 2, .Destination = 0, .MergeRecords = 1, \
    .From = 50, .To = 100, .MailMerge
```

MailMerge

¬Example

MailMerge [**.CheckErrors =** *number*] [**, .Destination =** *number*] [**, .MergeRecords =** *number*] [**, .From =** *number*] [**, .To =** *number*] [**, .Suppression =** *number*] [**, .MailMerge**] [**, .MailSubject =** *text*] [**, .MailAsAttachment =** *number*] [**, .MailAddress =** *text*]

Sets options for a mail merge, merges the main document with the specified data records, or both. If the active document is not a main document, an error occurs.

Argument	Explanation
.CheckErrors	Specifies how to check and report mail merge errors: 0 (zero) Simulates the mail merge and reports errors in a new document. 1 Performs a mail merge, pausing to report each error as it occurs. 2 Performs a mail merge and reports any errors in a new document.
.Destination	Specifies where to send the merge documents: 0 (zero) New document 1 Printer 2 Electronic mail; both the messaging application programming interface (MAPI) and vendor independent messaging (VIM) are supported. 3 Fax
.MergeRecords	Specifies whether or not to merge a subset of the data records: 0 (zero) Merges all data records. 1 Merges the range of data records specified by .From and .To.
.From	The number of the first data record to merge.
.To	The number of the last data record to merge.
.Suppression	Specifies whether or not to print blank lines for empty merge fields: 0 (zero) Does not print blank lines. 1 Prints blank lines.
.MailMerge	Performs the mail merge (omit .MailMerge if you want to set options only).
.MailSubject	If the destination is electronic mail, specifies the subject text.
.MailAsAttachment	If 1, and if the destination is electronic mail or fax, sends the merge documents as attachments.
.MailAddress	If the destination is electronic mail or fax, specifies the name of the merge field that contains the electronic mail address or fax

number.

See also

Mail Merge Statements and Functions

MailMergeCheck

MailMergeQueryOptions

MailMergeToDoc

MailMergeToPrinter

MailMergeAskToConvertChevrons, MailMergeAskToConvertChevrons()

MailMergeAskToConvertChevrons [*Prompt*]

MailMergeAskToConvertChevrons()

When you open a document created in Word for the Macintosh that contains chevrons (« »), the **MailMergeAskToConvertChevrons** statement controls whether or not Word displays a prompt asking if you want to convert text enclosed by chevrons to merge fields.

Argument	Explanation
Prompt	Specifies whether or not to display a prompt: 0 (zero) Does not display the prompt; whether Word converts merge fields is controlled by **MailMergeConvertChevrons**. 1 Displays the prompt. Omitted Toggles the option to display the prompt.

The **MailMergeAskToConvertChevrons()** function returns -1 if Word will display the prompt and 0 (zero) if it will not.

See also

Mail Merge Statements and Functions

MailMergeConvertChevrons

MailMergeCheck

MailMergeCheck .CheckErrors = *number*

Checks for errors and reports any that are found. You can check for errors with or without performing a mail merge.

Argument	Explanation
.CheckErrors	Specifies whether or not to perform a mail merge when checking for errors: 0 (zero) Simulates the mail merge and reports errors in a new document. 1 Performs a mail merge, pausing to report each error as it occurs. 2 Performs a mail merge and reports any errors in a new document.

See also

Mail Merge Statements and Functions

MailMerge

MailMergeConvertChevrons, MailMergeConvertChevrons()

MailMergeConvertChevrons [*Convert*]

MailMergeConvertChevrons()

When you open a document created in Word for the Macintosh that contains chevrons (« ») and **MailMergeAskToConvertChevrons()** returns 0 (zero), **MailMergeConvertChevrons** controls whether or not Word converts text enclosed by chevrons to merge fields. If **MailMergeAskToConvertChevrons()** returns 1, **MailMergeConvertChevrons** controls which button is the default when Word displays a prompt asking if you want to convert chevrons.

Argument	Explanation
Convert	Controls whether or not to convert chevrons (or which button is the default if the prompt appears): 0 (zero) Does not convert chevrons (or sets the default button in the prompt to No). 1 Converts chevrons (or sets the default button in the prompt to Yes). Omitted Toggles the setting that determines whether chevrons are converted (or toggles the default button in the prompt).

The **MailMergeConvertChevrons()** function returns 0 (zero) if the default button is No and -1 if the default button is Yes.

See also

Mail Merge Statements and Functions

MailMergeAskToConvertChevrons

MailMergeCreateDataSource

MailMergeCreateDataSource .FileName = *text* [**, .PasswordDoc** = *text*] [**, .HeaderRecord** = *text*] [**, .MSQuery**] [**, .SQLStatement** = *text*] [**, .SQLStatement1** = *text*] [**, .Connection** = *text*] [**, .LinkToSource** = *number*]

Creates a Word document that stores data for a mail merge. **MailMergeCreateDataSource** attaches the new data source to the active document, which becomes a main document if it is not one already.

To create a new data source with a table you can fill in later, specify only .FileName and .HeaderRecord (and .PasswordDoc if you want to protect the new data source).

Argument	Explanation
.FileName	The path and filename for the new data source
.PasswordDoc	A password to protect the new data source
.HeaderRecord	Field names for the header record. If omitted, the standard header record is used: "Title, FirstName, LastName, JobTitle, Company, Address1, Address2, City, State, PostalCode, Country, HomePhone, WorkPhone." To separate field names, use the character specified by the "sList" setting in the [intl] section of your WIN.INI file.

If you've installed Microsoft Query, you can retrieve a subset of data from a primary source --- for example, a Microsoft Excel worksheet --- using .MSQuery, .SQLStatement, and .Connection.

Argument	Explanation
.MSQuery	Uses Microsoft Query (if installed) to retrieve data specified by .SQLStatement and .Connection from the primary data source. Note that including .MSQuery without any other arguments simply displays Microsoft Query, if installed.
.SQLStatement	Defines query options for retrieving data. For information on valid query strings, see the Help file for the appropriate ODBC driver.
.SQLStatement1	If the query string is longer than 255 characters, .SQLStatement specifies the first portion of the string and .SQLStatement1 specifies the second portion.
.Connection	Specifies a range within which to perform the query specified by .SQLStatement. How you specify the range depends on how

	data is retrieved. For example: ¬ When retrieving data using ODBC, you specify a connection string. For information on valid connection strings, see the Help file for the appropriate ODBC driver. ¬ When retrieving data from Microsoft Excel using dynamic data exchange (DDE), you specify a named range. ¬ When retrieving data from Microsoft Access, you specify the word "Table" or "Query" followed by the name of a table or query.
.LinkToSource	If 1, establishes a link between the new data source and the primary data source.

Note

Help files for ODBC drivers are located in the SYSTEM subdirectory of your Windows directory. Filenames begin with the letters "DRV" and end with the .HLP filename extension (for example, DRVFOX.HLP and DRVACCSS.HLP).

See also

Mail Merge Statements and Functions

MailMergeCreateHeaderSource

MailMergeEditDataSource

MailMergeOpenDataSource

MailMergeCreateHeaderSource

MailMergeCreateHeaderSource [**.FileName =** *text*] [**.PasswordDoc =** *text*] [**, .HeaderRecord =** *text*]

Creates a Word document that stores a header record that is used in place of the data source header record in a mail merge. **MailMergeCreateHeaderSource** attaches the new header source to the active document, which becomes a main document if it is not one already.

Argument	Explanation
.FileName	The path and filename for the new header source
.PasswordDoc	A password to protect the new header source
.HeaderRecord	Field names for the header record. If omitted, the standard header record is used: "Title, FirstName, LastName, JobTitle, Company, Address1, Address2, City, State, PostalCode, Country, HomePhone, WorkPhone." To separate field names, use the character specified by the "sList" setting in the [intl] section of your WIN.INI file.

See also

Mail Merge Statements and Functions

MailMergeCreateDataSource

MailMergeEditHeaderSource

MailMergeOpenHeaderSource

MailMergeDataForm

MailMergeDataForm

Displays the Data Form dialog box for entering a new data record. You can use **MailMergeDataForm** in a main document, a data source, or any document containing data delimited by table cells or separator characters.

See also

Mail Merge Statements and Functions

MailMergeEditDataSource

MailMergeDataSource$()

MailMergeDataSource$(*Type*)

Returns information about the data or header source, the means of data retrieval, or the current connection or query string.

Type	Values and descriptions
0	The path and filename of the data source
1	The path and filename of the header source
2	A number, returned as text, indicating how data is being supplied for the mail merge operation: 0 From a Word document or through a Word file converter 1 Dynamic data exchange (DDE) from Microsoft Access 2 DDE from Microsoft Excel 3 DDE from Microsoft Query 4 Open database connectivity (ODBC)
3	A number, returned as text, indicating how the header source is being supplied for the mail merge operation. See values and descriptions for *Type* 2.
4	The connection string for the data source
5	The query string (SQL statement)

See also

Mail Merge Statements and Functions

MailMergeCreateDataSource

MailMergeEditDataSource

MailMergeOpenDataSource

MailMergeEditDataSource

MailMergeEditDataSource

Performs one of the following when a main document is active:

- If the data source is a Word document, opens the data source (or activates the data source if it is already open).
- If Word is accessing the data through dynamic data exchange (DDE) --- using an application such as Microsoft Excel, Microsoft Access, or Microsoft FoxPro --- displays the data source in that application.
- If Word is accessing the data through open database connectivity (ODBC), displays the data in a Word document. Note that if Microsoft Query is installed, a message appears providing the option to display Microsoft Query instead of converting data.

See also

Mail Merge Statements and Functions

MailMergeCreateDataSource

MailMergeEditMainDocument

MailMergeOpenDataSource

MailMergeEditHeaderSource

MailMergeEditHeaderSource

Opens the header source attached to the active main document or activates the header source if it is already open. If the active document is not a main document, or if a header source is not attached, an error occurs.

_____

See also

Mail Merge Statements and Functions

MailMergeCreateHeaderSource

MailMergeEditDataSource

MailMergeOpenHeaderSource

MailMergeEditMainDocument

MailMergeEditMainDocument

Activates the mail merge main document associated with the active header source or data source. If there is no main document associated with the active header or data source, or if the main document is not open, an error occurs.

_____

See also

Mail Merge Statements and Functions

MailMergeEditDataSource

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MailMergeFindRecord Example

This example displays a merge document for the first data record in which the last name is "Perez." If the data record is found, the number of the record is stored in the variable numRecord.

```
MailMergeFindRecord .Find = "Perez", .Field = "LastName"
If MailMergeFoundRecord() Then
    numRecord = MailMergeGotoRecord()
End If
```

MailMergeFindRecord

¬Example

MailMergeFindRecord .Find = *text***, .Field =** *text*

If a main document is active and merged data is visible, displays a merge document for the first data record containing the specified text in the given field. If a data source is active and is a Word document, **MailMergeFindRecord** selects the first row that matches the specified criteria. You can use **MailMergeFindRecord** in a main document, a data source, or any document containing data delimited by table cells or separator characters.

Argument	Explanation
.Find	Text you want to find in the specified field.
.Field	Limits the search to the specified field name.

See also

Mail Merge Statements and Functions

MailMergeFirstRecord

MailMergeFoundRecord()

MailMergeGotoRecord

MailMergeLastRecord

MailMergeNextRecord

MailMergePrevRecord

MailMergeViewData

MailMergeFirstRecord

MailMergeFirstRecord

If merged data is visible in the main document, displays a merge document for the first data record in the query result (or the first record in the data source if no query options are in effect). This statement is valid only when a main document is active.

_____

See also

Mail Merge Statements and Functions

MailMergeFindRecord

MailMergeGotoRecord

MailMergeLastRecord

MailMergeNextRecord

MailMergePrevRecord

MailMergeViewData

MailMergeFoundRecord()

MailMergeFoundRecord()

Returns a value indicating whether the most recent **MailMergeFindRecord** operation was successful.

Value	Explanation
-1	The find operation was successful.
0 (zero)	The find operation was not successful.

For an example, see MailMergeFindRecord Example.

See also

Mail Merge Statements and Functions

MailMergeFindRecord

MailMergeGotoRecord, MailMergeGotoRecord()

MailMergeGotoRecord *RecordNumber*

MailMergeGotoRecord()

If merged data is visible in the main document, the **MailMergeGotoRecord** statement displays a merge document for the data record corresponding to *RecordNumber.* Note that *RecordNumber* is the position of the record in the query result produced by the current query options, and is therefore not necessarily the position of the record in the data source.

The **MailMergeGotoRecord()** function returns the number of the data record currently displayed. If the active document is not a main document, both the statement and the function generate an error. For an example using **MailMergeGotoRecord()**, see MailMergeFindRecord Example.

See also

Mail Merge Statements and Functions

MailMergeFindRecord

MailMergeFirstRecord

MailMergeLastRecord

MailMergeNextRecord

MailMergePrevRecord

MailMergeViewData

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MailMergeHelper Example

This example displays the Mail Merge Helper dialog box.

```
Dim dlg As MailMergeHelper
GetCurValues dlg
x = Dialog(dlg)
```

MailMergeHelper

¬Example

MailMergeHelper

Used with a **Dialog** or **Dialog()** instruction to display the Mail Merge Helper dialog box (Mail Merge command, Tools menu), with which you can set up a main document, create and edit a data source, and perform a mail merge.

_____

See also

Mail Merge Statements and Functions

MailMerge

MailMergeInsertAsk

MailMergeInsertAsk .Name = *text* [**, .Prompt =** *text*] [**, .DefaultBookmarkText =** *text*] [**, .AskOnce =** *number*]

Inserts an ASK field into a main document at the insertion point. When updated, an ASK field displays a dialog box that prompts for text to assign to the specified bookmark.

Argument	Explanation
.Name	The name of the bookmark to which you want to assign the text typed in the dialog box.
.Prompt	A prompt that appears in the dialog box (for example, "Your Initials:").
.DefaultBookmarkText	Default text to assign to the bookmark.
.AskOnce	If 1, Word displays the prompt once at the beginning of the merge instead of once for each data record.

For more information, see ASK field.

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertFillIn

MailMergeInsertSet

MailMergeInsertFillIn

MailMergeInsertFillIn [**.Prompt =** *text*] [**, .DefaultFillInText =** *text*] [**, .AskOnce =** *number*]

Inserts a FILLIN field into a main document at the insertion point. When updated, a FILLIN field displays a dialog box that prompts for text to insert into the document at the location of the field.

Argument	Explanation
.Prompt	A prompt that appears in the dialog box (for example, "Your Initials:").
.DefaultFillInText	Default text to insert at the field location.
.AskOnce	If 1, Word displays the prompt once at the beginning of the merge operation instead of once for each data record.

For more information, see FILLIN field.

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertAsk

MailMergeInsertSet

MailMergeInsertIf

MailMergeInsertIf .MergeField = *text***, .Comparison =** *number* [**, .CompareTo =** *text*] [**, .TrueAutoText =** *text*] [**, .TrueText =** *text*] [**, .FalseAutoText =** *text*] [**, .FalseText =** *text*]

Inserts an IF field into a main document at the insertion point. When updated, an IF field compares a field in a data record to a specified value, and then inserts the appropriate text according to the result of the comparison.

Argument	Explanation
.MergeField	The name of the merge field you want to compare to the text specified by .CompareTo.
.Comparison	Specifies the operator for the comparison: 0 (zero) = (equal to) 1 <> (not equal to) 2 < (less than) 3 > (greater than) 4 <= (less than or equal to) 5 >= (greater than or equal to) 6 = " " (is blank) 7 <> " " (is not blank)
.CompareTo	Text to compare to the merge field. This argument is required unless .Comparison is set to 6 (is blank) or 7 (is not blank).
.TrueAutoText	An AutoText entry containing the text to insert if the comparison is true (.TrueText is ignored).
.TrueText	Text to insert if the comparison is true.
.FalseAutoText	An AutoText entry containing the text to insert if the comparison is false (.FalseText is ignored).
.FalseText	Text to insert if the comparison is false.

For more information, see IF field.

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertNext

MailMergeInsertNextIf

MailMergeInsertSkipIf

MailMergeInsertMergeRec

MailMergeInsertMergeRec

Inserts a MERGEREC field into a main document at the insertion point. A MERGEREC field inserts the number of the current data record (the position of the data record in the current query result) during a mail merge. For more information, see MERGEREC field.

_____

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertMergeSeq

MailMergeInsertMergeSeq

MailMergeInsertMergeSeq

Inserts a MERGESEQ field into a main document at the insertion point. A MERGESEQ field inserts a number based on the sequence in which data records are merged (for example, when merging records 50 to 100, inserts 1 when merging record number 50). For more information, see MERGESEQ field.

_____

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertMergeRec

MailMergeInsertNext

MailMergeInsertNext

Inserts a NEXT field into a main document at the insertion point. A NEXT field advances to the next data record so that data from more than one record can be merged into the same merge document (for example, a sheet of mailing labels). For more information, see NEXT field.

_____

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertNextIf

MailMergeInsertNextIf

MailMergeInsertNextIf .MergeField = *text*, **.Comparison =** *number*, **.CompareTo =** *text*

Inserts a NEXTIF field into a main document at the insertion point. If the comparison in the NEXTIF field is true, Word merges the next data record into those merge fields in the current merge document that follow the NEXTIF field. Otherwise, Word merges the next data record into a new merge document. In general, it is preferable to specify query options instead of using NEXTIF fields. For argument descriptions, see MailMergeInsertIf. For more information, see NEXTIF field.

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertIf

MailMergeInsertNext

MailMergeInsertSkipIf

MailMergeInsertSet

MailMergeInsertSet .Name = *text* [**, .ValueText =** *text*] [**, .ValueAutoText =** *text*]

Inserts a SET field into a main document at the insertion point. A SET field defines the text of the specified bookmark.

Argument	Explanation
.Name	The name of the bookmark to define
.ValueText	Text to assign the bookmark to
.ValueAutoText	The AutoText entry whose text you want to assign the bookmark to (.ValueText is ignored)

For more information, see SET field.

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertAsk

MailMergeInsertFillIn

MailMergeInsertSkipIf

MailMergeInsertSkipIf .MergeField = *text* **, .Comparison =** *number* [**, .CompareTo =** *text*]

Inserts a SKIPIF field into a main document at the insertion point. If the comparison in the SKIPIF field is true, Word cancels the current merge document and skips to the next data record. In general, it is preferable to specify query options instead of using SKIPIF fields. For argument descriptions, see MailMergeInsertIf. For more information, see SKIPIF field.

See also

Mail Merge Statements and Functions

InsertField

MailMergeInsertIf

MailMergeInsertNext

MailMergeInsertNextIf

MailMergeLastRecord

MailMergeLastRecord

If merged data is visible in the main document, displays a merge document for the last data record in the current query result (or the last record in the data source if no query options are in effect). This statement is valid only when a main document is active.

_____

See also

Mail Merge Statements and Functions

MailMergeFindRecord

MailMergeFirstRecord

MailMergeGotoRecord

MailMergeNextRecord

MailMergePrevRecord

MailMergeViewData

MailMergeMainDocumentType, MailMergeMainDocumentType()

MailMergeMainDocumentType *Type*

MailMergeMainDocumentType()

The **MailMergeMainDocumentType** statement makes the active window a main document. Note that if the active document is already a main document, Word removes the attached data source, if any.

Argument	Explanation
Type	The type of merge documents you will create with the main document: 0 (zero) or omitted Form letters 1 Mailing labels 2 Envelopes 3 Catalog documents

The **MailMergeMainDocumentType()** function returns one of the *Type* values if the active document is a main document, or -1 if it is a regular Word document.

See also

Mail Merge Statements and Functions

MailMergeCreateDataSource

MailMergeOpenDataSource

MailMergeReset

MailMergeNextRecord

MailMergeNextRecord

If merged data is visible in the main document, displays a merge document for the next record in the current reply (or the next record in the data source if no query options are in effect). This statement is valid only when a main document is active.

_____

See also

Mail Merge Statements and Functions

MailMergeFindRecord

MailMergeFirstRecord

MailMergeGotoRecord

MailMergeLastRecord

MailMergePrevRecord

MailMergeViewData

MailMergeOpenDataSource

MailMergeOpenDataSource .Name = *text* [**, .ConfirmConversions =** *number*] [**, .ReadOnly** = *number*] [**, .LinkToSource =** *number*] [**, .AddToMru =** *number*] [**, .PasswordDoc** = *text*] [**, .PasswordDot** = *text*] [**, .Revert =** number] [**, .WritePasswordDoc =** *text*] [**, .WritePasswordDot =** *text*] [**, .Connection =** *text*] [**, .SQLStatement =** *text*] [**, .SQLStatement1 =** *text*]

Attaches the specified data source to the active document, which becomes a main document if it is not one already.

Argument	Explanation
.Name	The name of the data source. Note that you can specify a Microsoft Query (.QRY) file instead of a specifying a data source, a connection string, and a query string.
.LinkToSource	If 1, the query specified by .Connection and .SQLStatement is performed each time the main document is opened.
.Connection	Specifies a range within which to perform the query specified by .SQLStatement. How you specify the range depends on how data is retrieved. For example: ¬ When retrieving data through ODBC, you specify a connection string. For information on valid connection strings, see the Help file for the appropriate ODBC driver. ¬ When retrieving data from Microsoft Excel using dynamic data exchange (DDE), you specify a named range. ¬ When retrieving data from Microsoft Access, you specify the word "Table" or "Query" followed by the name of a table or query.
.SQLStatement	Defines query options for retrieving data. For information on valid query strings, see the Help file for the appropriate ODBC driver.
.SQLStatement 1	If the query string is longer than 255 characters, .SQLStatement specifies the first portion of the string and .SQLStatement1 specifies the second portion.

Note

Help files for ODBC drivers are located in the SYSTEM subdirectory of your Windows directory. Filenames begin with the letters "DRV" and end with the .HLP filename extension (for example, DRVFOX.HLP and DRVACCSS.HLP).

For descriptions of other arguments, see FileOpen.

See also

Mail Merge Statements and Functions

FileOpen

MailMergeCreateDataSource

MailMergeEditDataSource

MailMergeOpenHeaderSource

MailMergeOpenHeaderSource

MailMergeOpenHeaderSource .Name = *text* [**, .ConfirmConversions =** *number*] [**, .ReadOnly** = *number*] [**, .AddToMru =** *number*] [**, .PasswordDoc** = *text*] [**, .PasswordDot** = *text*] [**, .Revert =** number] [**, .WritePasswordDoc =** *text*] [**, .WritePasswordDot =** *text*]

Attaches the specified header source to the active document, which becomes a main document if it is not one already. The header record in the header source is used in place of the header record in the data source. For argument descriptions, see FileOpen.

See also

Mail Merge Statements and Functions

MailMergeCreateHeaderSource

MailMergeEditHeaderSource

MailMergeOpenDataSource

MailMergePrevRecord

MailMergePrevRecord

If merged data is visible in the main document, displays a merge document for the previous record in the current query result (or the previous record in the data source if no query options are in effect). This statement is valid only when a main document is active.

_____

See also

Mail Merge Statements and Functions

MailMergeFindRecord

MailMergeFirstRecord

MailMergeGotoRecord

MailMergeLastRecord

MailMergeNextRecord

MailMergeQueryOptions

MailMergeQueryOptions .SQLStatement = *text* [**.SQLStatement1 =** *text*]

Specifies the query options for a mail merge. You can use **MailMergeQueryOptions** to change the query options established with a **MailMergeCreateDataSource** or **MailMergeOpenDataSource** instruction. If the active document is not a main document, an error occurs.

Argument	Explanation
.SQLStatement	An optional query string that retrieves a subset of the data in a primary data source. For information on valid query strings, see the Help file for the appropriate ODBC driver.
.SQLStatement 1	If the query string is longer than 255 characters, .SQLStatement specifies the first portion of the string and .SQLStatement1 specifies the second portion.

Note

Help files for ODBC drivers are located in the SYSTEM subdirectory of your Windows directory. Filenames begin with the letters "DRV" and end with the .HLP filename extension (for example, DRVFOX.HLP and DRVACCSS.HLP).

See also

Mail Merge Statements and Functions

MailMergeCreateDataSource

MailMergeOpenDataSource

MailMergeReset

MailMergeReset

Detaches the data and header sources from the main document and resets it to a regular Word document. If the active document is not a main document, an error occurs.

See also

Mail Merge Statements and Functions

MailMergeMainDocumentType

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MailMergeState() Example

This example checks the state of the active document before performing a mail merge. If the active document is not a main document, Word displays a message box. If the document is a main document but has no attached data source, a data source is attached. In any other case, the mail merge begins immediately.

```
Select Case MailMergeState(0)
    Case 0
        MsgBox "Not a mail merge main document."
        quitmacro = 1
    Case 1, 3
        MailMergeOpenDataSource .Name = "C:\DATA\DATA.DOC"
    Case Else
End Select
If quitmacro = 1 Then Goto bye
MailMergeToPrinter
bye:
```

MailMergeState()

¬Example

MailMergeState(*Type*)

Returns one of four different types of information about the current state of a mail merge setup. *Type* 1, *Type* 2, and *Type* 3 return -1 if the active document is not a main document, data source, or header source.

Type	Values and descriptions
0 (zero)	Returns information about the active document: 0 (zero) Regular Word document 1 Main document with no data or header source attached 2 Main document with an attached data source 3 Main document with an attached header source 4 Main document with both a data and header source attached 5 Data source or header source; associated main document is open
1	Returns the kind of main document: 0 (zero) Form letters 1 Mailing labels 2 Envelopes 3 Catalog
2	Returns information about the selected mail merge options: 0 (zero) Neither blank-line suppression nor query options are enabled. 1 Blank-line suppression is enabled. 2 Query options are enabled. 3 Both blank-line suppression and query options are enabled.
3	Returns the mail merge destination: 0 (zero) New document 1 Printer 2 Electronic mail 3 Fax

See also

Mail Merge Statements and Functions

MailMergeMainDocumentType

MailMergeToDoc

MailMergeToDoc

Merges data records with the main document and sends the resulting merge documents to a new document. If the most recent mail merge specified a range of data records to merge, only those records are merged.

_____

See also

Mail Merge Statements and Functions

MailMerge

MailMergeToPrinter

MailMergeToPrinter

MailMergeToPrinter

Merges data records with the main document and prints the resulting merge documents. If the most recent mail merge specified a range of data records to merge, only those records are merged.

_____

See also

Mail Merge Statements and Functions

MailMerge

MailMergeToDoc

MailMergeViewData, MailMergeViewData()

MailMergeViewData [*DisplayResults*]

MailMergeViewData()

The **MailMergeViewData** statement controls the display of merge fields in a main document. If the active document is not a main document, an error occurs.

Argument	Explanation
DisplayResults	Specifies whether to display merge field names in chevrons (« ») or information from the current data record in place of the merge fields: 0 (zero) Merge field names 1 Information from the current data record

The **MailMergeViewData()** function returns 0 (zero) if merge field names are displayed and 1 if information from the current data record is displayed.

See also

Mail Merge Statements and Functions

MailMergeEditDataSource

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

MergeFieldName$() Example

This example defines an array containing the field names available to a main document (which must be active when the example is run):

```
numFields = CountMergeFields()
Dim mmFields$(numFields - 1)
For count = 1 To numFields
    mmFields$(count - 1) = MergeFieldName$(count)
Next count
```

MergeFieldName$()

¬Example

MergeFieldName$(*Count***)**

Returns a field name in a data source or header source. In a data source set up as a Word table, the field names are in the first row as column headings. In a data source set up as delimited lists, the field names are in the first paragraph. An error occurs if you run **MergeFieldName$()** when the active document is not a main document.

Argument	Explanation
Count	The number corresponding to the field name in the data source or header source, where 1 is the first field name, 2 is the second, and so on.

See also

Mail Merge Statements and Functions

CountMergeFields()

InsertMergeField

ToolsAddRecordDefault

ToolsAddRecordDefault

Adds an empty record to the end of a data source. For example, **ToolsAddRecordDefault** adds an empty row to the bottom of a table in a Word document containing data for a mail merge. **ToolsAddRecordDefault** can be used with any document that could be used as a data source during a mail merge.

_____

See also

Mail Merge Statements and Functions

MailMergeEditDataSource

ToolsRemoveRecordDefault

ToolsRemoveRecordDefault

ToolsRemoveRecordDefault

Removes the data record containing the insertion point --- for example, removes the table row containing the insertion point. **ToolsRemoveRecordDefault** can be used with any document that could be used as a data source during a mail merge.

_____

See also

Mail Merge Statements and Functions

MailMergeEditDataSource

ToolsAddRecordDefault

AtEndOfDocument()

AtEndOfDocument()

Returns -1 if the insertion point is at the end of the document or 0 (zero) if it is not. Unlike the **EndOfDocument()** function, **AtEndOfDocument()** does not move the insertion point.

See also

Moving the Insertion Point and Selecting Statements and Functions

AtStartOfDocument()

EndOfDocument

AtStartOfDocument()

AtStartOfDocument()

Returns -1 if the insertion point is at the beginning of the document or 0 (zero) if it is not. Unlike the **StartOfDocument()** function, **AtStartOfDocument()** does not move the insertion point.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

AtEndOfDocument()

StartOfDocument

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CharLeft Examples

This example moves the insertion point five characters to the left. The **If** conditional determines whether there is a selection. If there is a selection, the CharLeft 1 instruction changes the selection to an insertion point at the left end of the selection before moving the insertion point five characters to the left. This ensures that the insertion point is moved five characters whether or not there is a selection.

```
If SelType() = 2 Then
    CharLeft 1
    CharLeft 5
Else
    CharLeft 5
End If
```

The following example selects the current paragraph and then shrinks the selection by one character, so that the paragraph text is selected, but not the paragraph mark. You might want to do this if you need to copy the text of a paragraph but not its paragraph formatting, which is stored in the paragraph mark.

```
EditGoTo "\Para"
CharLeft 1, 1
```

CharLeft, CharLeft()

¬Example

CharLeft [*Count*] [**,** *Select*]

CharLeft([*Count*] [**,** *Select*]**)**

The **CharLeft** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+LEFT ARROW) to the left by the specified number of characters.

Argument	Explanation
Count	The number of characters to move. If omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection to the right.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **CharLeft** moves the insertion point *Count*-1 characters to the left of the selection. Nonzero Text is selected. If there is already a selection, **CharLeft** moves the active end of the selection toward the beginning of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **CharLeft** shrinks the selection. In a selection made from right to left, it extends the selection.

If there is a selection, CharLeft 1 changes the selection to an insertion point positioned at the left end of the original selection.

The **CharLeft()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved to the left.
-1	If the insertion point or the active end of the selection is moved to the left by any number of characters, even if less than *Count*. For example, CharLeft(10) returns -1 even if the insertion point is only three characters from the start of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

CharRight

SentLeft

SentRight

WordLeft
WordRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CharRight Examples

This example moves the insertion point to the start of the current sentence and then selects the first five characters to the right:

```
SentLeft
CharRight 5, 1
```

The following example first extends the selection five characters to the right, copies the selection, and then moves the insertion point five characters to the right of the selection. Note that CharRight 6 is used to move the selection five characters to the right of the selection. This instruction is equivalent to CharRight 1, which moves the insertion point to the right end of the selection, plus CharRight 5, which then moves it five characters.

```
CharRight 5, 1
EditCopy
CharRight 6
```

CharRight, CharRight()

¬Example

CharRight [*Count*] [**,** *Select*]

CharRight([*Count*] [**,** *Select*]**)**

The **CharRight** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+RIGHT ARROW) to the right by the specified number of characters.

Argument	Explanation
Count	The number of characters to move. If omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection to the left.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **CharRight** moves the insertion point *Count*-1 characters to the right of the selection. Nonzero Text is selected. If there is already a selection, **CharRight** moves the active end of the selection toward the end of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **CharRight** extends the selection. In a selection made from right to left, it shrinks the selection.

If there is a selection, CharRight 1 changes the selection to an insertion point positioned at the right end of the original selection.

The **CharRight()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved to the right.
-1	If the insertion point or the active end of the selection is moved to the right by any number of characters, even if less than *Count*. For example, CharRight(10) returns -1 even if the insertion point is only three characters from the end of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

CharLeft

SentLeft

SentRight

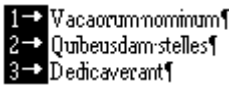
WordLeft
WordRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ColumnSelect Example

This example selects the numbers and tab characters in the current line and the two subsequent lines of a list, as in the illustration that follows it:

```
StartOfLine
ColumnSelect
WordRight 2
LineDown 2
```



ColumnSelect

¬Example

ColumnSelect

Turns on column selection mode to select a column of text, such as numbers aligned at the same tab stop in two or more lines of text. **Cancel** or any command acting on the column selection ends this mode. **ColumnSelect** does not select a column in a Word table; use **TableSelectColumn** for this purpose.

See also

Moving the Insertion Point and Selecting Statements and Functions

Cancel

ExtendSelection

EditSelectAll

EditSelectAll

Selects the entire document.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EndOfColumn Example

This example moves the insertion point from anywhere in the table to the beginning of the last cell in the last row:

```
EndOfRow
EndOfColumn
```

EndOfColumn, EndofColumn()

¬Example

EndOfColumn [*Select*]

EndOfColumn ([*Select*]**)**

The **EndOfColumn** statement moves the insertion point or extends the selection (if *Select* is nonzero) to the bottom of the table column containing the insertion point or selection. If the selection extends over more than one column, the insertion point moves or the selection is extended to the bottom of the rightmost column in the selection. If the insertion point or selection is not in a table, an error occurs.

Note

If the last row in the table does not have a cell that corresponds to the column that contains the current selection --- for example, if you have deleted or merged cells in the last row --- **EndOfColumn** moves the insertion point or extends the selection to the end of the last row in the table.

The **EndOfColumn()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point was not moved or the selection was not extended (that is, if it was already at the bottom of the column)
-1	If the insertion point was moved or the selection was extended

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfRow

StartOfColumn

EndOfDocument, EndOfDocument()

EndOfDocument [*Select*]

EndOfDocument([*Select*]**)**

The **EndOfDocument** statement moves the insertion point or, if *Select* is nonzero, the active end of the selection (the end that moves when you press CTRL+SHIFT+END) to the end of the document.

The **EndOfDocument()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection was not moved (for example, if it was already at the end of the document)
-1	If the insertion point or the active end of the selection was moved

See also

Moving the Insertion Point and Selecting Statements and Functions

AtEndOfDocument()

StartOfDocument

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EndOfLine Examples

This example selects the current line and the paragraph mark (if there is one at the end of the line). The instruction EditGoTo "\Line" has the same effect.

```
StartOfLine
EndOfLine 1
```

The following example selects the current line but not the paragraph mark (if there is one at the end of the line):

```
EndOfLine
StartOfLine 1
```

EndOfLine, EndOfLine()

¬Example

EndOfLine [*Select*]

EndOfLine ([*Select*]**)**

The **EndOfLine** statement moves the insertion point or, if *Select* is nonzero, the active end of the selection (the end that moves when you press SHIFT+END) to the end of the current line or the line that contains the active end of the selection. If there is a paragraph mark at the end of the line, EndOfLine positions the insertion point to the left of the paragraph mark; EndOfLine 1 moves the active end of the selection over the paragraph mark.

The **EndOfLine()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection was not moved (that is, if it was already at the end of the line)
-1	If the insertion point or the active end of the selection was moved

Avoid using **EndOfLine** by itself to go to the end of a paragraph unless you are sure that the paragraph is a single line (for example, a word in a list of words). Instead, use the following instructions:

```
ParaDown        'Go to start of next paragraph
CharLeft        'Go back one character
```

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfRow

ParaDown

StartOfLine

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EndOfRow Example

This example deletes the selected cells or the cell containing the insertion point and all cells to the right:

```
EndOfRow 1
TableDeleteCells
```

EndOfRow, EndOfRow()

¬Example

EndOfRow [*Select*]

EndOfRow ([*Select*]**)**

The **EndOfRow** statement moves the insertion point or extends the selection (if *Select* is nonzero) to the last cell of the table row containing the insertion point or selection. If the selection extends over more than one row, the insertion point moves or the selection is extended to the last cell of the last row in the selection. If the insertion point or selection is not in a table, an error occurs.

The **EndOfRow()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point was not moved or the selection was not extended (that is, if it was already at the end of the row)
-1	If the insertion point was moved or the selection was extended

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfColumn

EndOfLine

StartOfRow

EndOfWindow, EndOfWindow()

EndOfWindow [*Select*]

EndOfWindow([*Select*]**)**

The **EndOfWindow** statement moves the insertion point or, if *Select* is nonzero, the active end of the selection (the end that moves when you press CTRL+SHIFT+PAGE DOWN) to the lower-right corner of the contents currently visible in the document window.

The **EndOfWindow()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection was not moved (that is, if it was already at the lower-right corner of the window)
-1	If the insertion point or the active end of the selection was moved

For an example, see StartOfWindow Example.

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfDocument

StartOfWindow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ExtendSelection Example

This example selects all text from the insertion point to and including the bookmark "MyMark":

```
ExtendSelection
EditGoTo "MyMark"
Cancel
```

ExtendSelection

¬Example

ExtendSelection [*Character$*]

Performs one of the following actions:

- If extend mode is off, activates extend mode (in extend mode, actions that normally move the insertion point move the active end of the selection).
- If extend mode is already on, extends the selection to the next unit of text (the progression is as follows: word, sentence, paragraph, section, entire document).
- If *Character$* is specified, extends the selection from the insertion point or the fixed end of the current selection (the end of the selection that does not move when you press SHIFT+an arrow key) to the next instance of that character without activating or deactivating extend mode.

You can use **ExtendSelection** with **EditGoTo** and **EditFind** to extend the selection to a specific location or piece of text. Use the **Cancel** statement to deactivate extend mode.

See also

Moving the Insertion Point and Selecting Statements and Functions

Cancel

ColumnSelect

ExtendMode()

ShrinkSelection

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetSelEndPos() Example

This macro displays a message box if there is no selection. The user-defined function checksel compares the character positions at the start and end of the selection. If the positions are equal, indicating there is no selection, the function returns 0 (zero) and the message box is displayed.

```
Sub MAIN
    If checksel = 0 Then MsgBox "There is no selection."
End Sub


Function checksel
    If GetSelStartPos() = GetSelEndPos() Then
        checksel = 0
    Else
        checksel = 1
    End If
End Function
```

GetSelEndPos()

¬Example

GetSelEndPos()

Returns the character position of the end of the selection relative to the start of the document, which has a character position of 0 (zero). The end of the selection is defined as the end farthest from the start of the document. All characters, including nonprinting characters, are counted. Hidden characters are counted even if they are not displayed.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

GetSelStartPos()

GetText$()

SetSelRange

GetSelStartPos()

GetSelStartPos()

Returns the character position of the start of the selection relative to the start of the document, which has a character position of 0 (zero). The start of the selection is defined as the end closest to the start of the document. All characters, including nonprinting characters, are counted. Hidden characters are counted even if they are not displayed.

For an example, see GetSelEndPos() Example.

See also

Moving the Insertion Point and Selecting Statements and Functions

GetSelEndPos()

GetText$()

SetSelRange

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GetText$() Example

This example sets the variable a$ to the first 10 characters in the document:

```
a$ = GetText$(0, 10)
```

GetText$()

¬Example

GetText$(*CharPos1***,** *CharPos2***)**

Returns the text (unformatted) from *CharPos1* through *CharPos2*. Note that **GetText$()** does not return hidden text that is not displayed even though Word counts hidden characters when determining the range of text to return. Though **GetText$()** can be handy if you need to return document text without changing the selection, special document elements such as fields, tables, and page breaks within the specified range can produce unpredictable results.

Argument	Explanation
CharPos1	The character position that defines the beginning of the range, relative to the start of the document, which has a character position of 0 (zero)
CharPos2	The character position that defines the end of the range

See also

Moving the Insertion Point and Selecting Statements and Functions

GetSelEndPos()

GetSelStartPos()

SetSelRange

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

GoBack Example

An AutoExec macro such as the following runs each time you start Word. If there is at least one document listed at the bottom of the File menu, this macro opens the document and moves the insertion point to the most recent editing location.

```
Sub MAIN
If FileName$(1) <> "" Then
    FileList 1
    GoBack
End If
End Sub
```

GoBack

¬Example

GoBack

Moves the insertion point among the last four locations where editing occurred (the same as pressing SHIFT+F5).

See also

Moving the Insertion Point and Selecting Statements and Functions

EditGoTo

GoToHeaderFooter

GoToHeaderFooter

Moves the insertion point from a header to a footer, or vice versa. If the insertion point is not in a header or footer, an error occurs.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

CloseViewHeaderFooter

ShowNextHeaderFooter

ShowPrevHeaderFooter

ViewFooter

ViewHeader

GoToNext*Item*

GoToNext*Item*

Moves the insertion point to the next item specified by *Item*.

Argument	Explanation
Item	The item you want to go to. The following statements are available: **GoToNextAnnotation** **GoToNextEndnote** **GoToNextFootnote** **GoToNextPage** **GoToNextSection** **GoToNextSubdocument**

See also

Moving the Insertion Point and Selecting Statements and Functions

EditGoTo

GoToHeaderFooter

GoToPrevious*Item*

GoToPrevious*Item*

GoToPrevious*Item*

Moves the insertion point to the previous item specified by *Item*.

Argument	Explanation
Item	The item you want to go to. The following statements are available: **GoToPreviousAnnotation** **GoToPreviousEndnote** **GoToPreviousFootnote** **GoToPreviousPage** **GoToPreviousSection** **GoToPreviousSubdocument**

See also

Moving the Insertion Point and Selecting Statements and Functions

EditGoTo

GoBack

GoToNext*Item*

HLine

HLine [*Count*]

Scrolls the active document horizontally. A "line" corresponds to clicking a scroll arrow on the horizontal scroll bar once.

Argument	Explanation
Count	The amount to scroll, in lines: Omitted — One line to the right > 0 (zero) — The specified number of lines to the right < 0 (zero) — The specified number of lines to the left; scrolls into the left margin only if there is text in the margin

See also

Moving the Insertion Point and Selecting Statements and Functions

HPage

HScroll

VLine

HPage

HPage [*Count*]

Scrolls the active document horizontally. **HPage** corresponds to clicking the horizontal scroll bar to the left or right of the scroll box.

Argument	Explanation
Count	The amount to scroll, in document-window widths: Omitted — One width to the right > 0 (zero) — The specified number of widths to the right < 0 (zero) — The specified number of widths to the left

See also

Moving the Insertion Point and Selecting Statements and Functions

HLine

HScroll

VPage

HScroll, HScroll()

HScroll *Percentage*

HScroll()

The **HScroll** statement scrolls horizontally to the specified percentage of the document width. **HScroll** corresponds to dragging the scroll box on the horizontal scroll bar.

The **HScroll()** function returns the current horizontal scroll position as a percentage of the document width. **HScroll()** does not return negative values; if the document is scrolled into the left margin, **HScroll()** returns 0 (zero).

See also

Moving the Insertion Point and Selecting Statements and Functions

HLine

HPage

VScroll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Insert Examples

This example inserts the text "Hamlet" (without quotation marks) at the insertion point:

```
Insert "Hamlet"
```

The following example inserts the text "Hamlet" enclosed in quotation marks at the insertion point:

```
Insert Chr$(34) + "Hamlet" + Chr$(34)
```

This example inserts the text "GroupA:35" at the insertion point. To create the number portion of the string, **Str$()** converts a numeric variable to a string, and **LTrim$()** removes the leading space that is automatically added by **Str$()**.

```
num = 35
num$ = LTrim$(Str$(num))
Insert "GroupA:" + num$
```

Insert

¬Example

Insert *Text$*

Inserts the specified text at the insertion point. You can insert characters such as quotation marks, tab characters, nonbreaking hyphens, and newline characters using the **Chr$()** function with **Insert**. You can insert numbers by first converting them to text with the **Str$()** function.

See also

Moving the Insertion Point and Selecting Statements and Functions

Chr$()

InsertPara

LTrim$()

Str$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

LineDown Examples

This example moves the insertion point down five lines:

```
LineDown 5
```

The following example uses **LineDown()** to count the lines in a document:

```
StartOfDocument
lines = 1
While LineDown()
    lines = lines + 1
Wend
MsgBox "There are" + Str$(lines) + " lines in the document."
```

LineDown, LineDown()

¬Example

LineDown [*Count*] [**,** *Select*]

LineDown([*Count*] [**,** *Select*]**)**

The **LineDown** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+DOWN ARROW) down by the specified number of lines.

Argument	Explanation
Count	The number of lines to move down; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection up.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **LineDown** moves the insertion point *Count* lines below the selection. Nonzero Text is selected. If there is already a selection, **LineDown** moves the active end of the selection toward the end of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **LineDown** extends the selection. In a selection made from right to left, it shrinks the selection.

The **LineDown()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved down.
-1	If the insertion point or the active end of the selection is moved down by any number of lines, even if less than *Count*. For example, LineDown(10) returns -1 even if the insertion point is only three lines above the end of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

LineUp

ParaDown

ParaUp

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

LineUp Examples

This example moves the insertion point up 20 lines:

```
LineUp 20
```

The following example displays a message box if the insertion point is in the first line of the document:

```
If LineUp() = 0 Then MsgBox "No lines above here."
```

LineUp, LineUp()

¬Example

LineUp [*Count*] [**,** *Select*]

LineUp([*Count*] [**,** *Select*]**)**

The **LineUp** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+UP ARROW) up by the specified number of lines.

Argument	Explanation
Count	The number of lines to move up; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection down.
Select	Specifies whether to select text: 0 (zero) or omitted — Text is not selected. If there is already a selection, **LineUp** moves the insertion point *Count* lines above the selection. Nonzero — Text is selected. If there is already a selection, **LineUp** moves the active end of the selection toward the beginning of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **LineUp** shrinks the selection. In a selection made from right to left, it extends the selection.

The **LineUp()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved up.
-1	If the insertion point or the active end of the selection is moved up by any number of lines, even if less than *Count*. For example, LineUp(10) returns -1 even if the insertion point is three lines below the start of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

LineDown

ParaDown

ParaUp

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

NextCell Examples

This example moves the selection to the next cell, and then uses **SelType()** to determine if the cell is empty. If **SelType()** returns 1 (the value for the insertion point, indicating the cell is empty), a message box appears.

```
NextCell
If SelType() = 1 Then MsgBox "Empty cell!"
```

The following example uses the **NextCell()** function to determine if the insertion point is in the last cell. If the current cell is the last cell, the entire table is selected. If the current cell is not the last cell, the next cell is selected.

```
If NextCell() = 0 Then TableSelectTable
```

NextCell, NextCell()

¬Example

NextCell

NextCell()

The **NextCell** statement selects the contents of the next table cell (the same as pressing TAB in a table). If more than one cell is selected, **NextCell** selects the contents of the first cell in the selection. If the insertion point or selection is in the last cell of the table, **NextCell** adds a new row.

The **NextCell()** function behaves the same as the statement except when the insertion point or selection is completely within the last cell. In that case, the function returns 0 (zero) without enlarging the section, whereas the statement adds a new row.

Note

You can use EditGoTo .Destination = "\Cell" to select the contents of the current cell. The predefined bookmark "\Cell" and a number of others are set and updated automatically. For information on predefined bookmarks, see Predefined Bookmarks.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

PrevCell

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

NextField Example

This example counts the number of fields in the document (excluding XE, TA, TC, and RD fields) and displays the result in a message box. Note that the **While...Wend** loop does not include a **NextField** statement because the While NextField() instruction also moves the selection.

```
StartOfDocument
count = 0
While NextField()
    count = count + 1
Wend
MsgBox "Number of fields in the document:" + Str$(Count)
```

NextField, NextField()

¬Example

NextField

NextField()

The **NextField** statement selects the next field, regardless of whether the field is showing its codes or results. **NextField** skips over the following fields, which are formatted as hidden text: XE (Index Entry), TA (Table of Authorities Entry), TC (Table of Contents Entry), and RD (Referenced Document).

The **NextField()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If there is no next field (in other words, if the selection is not moved)
1	If the selection is moved

With field codes displayed, you can use **EditFind** to go to the next field of any type, including XE, TA, TC, and RD fields, provided that hidden text is displayed. Specify "^d" (the code for a field character) as the text for the .Find argument, as shown in the following instructions:

```
ViewFieldCodes 1
EditFind .Find = "^d", .Direction = 0, .Format = 0
```

To find only XE fields, specify "^d XE" as the text to find.

See also

Moving the Insertion Point and Selecting Statements and Functions

PrevField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

NextObject Example

This example moves the insertion point to the first document object on the current page. The EditGoTo "\Page" instruction selects the current page, and SelType 1 cancels the selection, leaving the insertion point at the top of the page. The NextObject instruction moves the insertion point to the first document object.

```
ViewPage
EditGoTo "\Page"
SelType 1
NextObject
```

NextObject

¬Example

NextObject

Moves the insertion point to the next document object on the current page in page layout view (the same as pressing ALT+DOWN ARROW). Document objects include text columns, table cells, footnotes, and frames.

From the top of the first text column on a page, the **NextObject** statement moves through document objects in the following order:

- Each text column, in order from first to last (if there is more than one), moving through all table cells and then all footnotes that appear within the text column
- Frames, if any appear, in the order that they appear in normal view, from the previous page break to the following page break

NextObject finally moves to the top of the first text column before cycling through the document objects again.

Keep the following points in mind when using **NextObject**:

- You can identify document objects by selecting the Text Boundaries check box, on the View tab in the Options dialog box (Tools menu), when the active document is in page layout view.
- Word does not consider OLE objects, such as embedded drawings and charts, to be document objects unless they are framed.
- If the insertion point is in a header or footer, **NextObject** moves between the header and footer.
- Make sure to switch to page layout view before using **NextObject**.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

PrevObject

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

NextPage Example

This example scrolls forward one page and positions the insertion point at the top of the document window. If the document has scrolled as far as possible, a message box is displayed.

```
ViewPage
If NextPage() = 0 Then
    MsgBox "Can't scroll!"
Else
    StartOfWindow
End If
```

NextPage, NextPage()

¬Example

NextPage

NextPage()

The **NextPage** statement scrolls forward one page in page layout view without moving the insertion point (the same as clicking the Page Forward button at the bottom of the vertical scroll bar in page layout view). If you want to move the insertion point after scrolling, include a **StartOfWindow** statement following **NextPage** in your macro.

NextPage scrolls from the current location on one page to the same relative location on the next page. To scroll to the beginning of the next page regardless of what portion of the current page appears, use the instruction EditGoTo .Destination = "p" instead of **NextPage**. Note, however, that **EditGoTo** also moves the insertion point.

The **NextPage()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If there is no next page (in other words, if the document doesn't scroll)
1	If the document scrolls

Note

NextPage and **NextPage()** are available only in page layout view and generate an error if run in another view.

See also

Moving the Insertion Point and Selecting Statements and Functions

EditGoTo

PageDown

PrevPage

ViewPage

VPage

NextWindow

NextWindow

Activates the window listed immediately after the active window on the Window menu. If the active window is last on the list, **NextWindow** activates the window at the top of the list. This statement does nothing if the active window is the only open document window.

Note

Window names are listed on the Window menu in alphabetic order. A path is included in a window name if the file is stored in a directory other than the current directory. When you change the current directory --- for example, by selecting a new directory in the Open dialog box (File menu) --- the window names may change, causing Word to re-sort them alphabetically.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

Activate

ChDir

PrevWindow

Window()

WindowList

WindowName$()

Window*Number*

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PageDown Examples

This example extends the selection down two screens from the insertion point:

```
PageDown 2, 1
```

The following example counts the number of screens between the insertion point and the end of the document. The **PageDown()** function is used to detect when the end of the document is reached.

```
n = 0
While PageDown()
    n = n + 1
Wend
MsgBox "Number of screens:" + Str$(n)
```

PageDown, PageDown()

¬Example

PageDown [*Count*] [**,** *Select*]

PageDown([*Count*] [**,** *Select*]**)**

The **PageDown** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+PAGE DOWN) down by the specified number of screens, where one screen is equal to the height of the active window. If there is not a full screen between the insertion point or the selection and the end of the document, **PageDown** moves the insertion point or the active end of the selection to the end of the document.

Argument	Explanation
Count	The number of screens to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection up.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **PageDown** moves the insertion point *Count*-1 screens below the selection. Nonzero Text is selected. If there is already a selection, **PageDown** moves the active end of the selection toward the end of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **PageDown** extends the selection. In a selection made from right to left, it shrinks the selection.

The **PageDown()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved down.
-1	If the insertion point or the active end of the selection is moved down by any number of screens, even if less than *Count*. For example, PageDown(2) returns -1 even if the insertion point is only one screen from the end of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

EditGoTo

HPage

HScroll

NextPage

PageUp

VPage
VScroll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PageUp Example

This example selects one screen up from the insertion point. If there is less than one screen above the insertion point, the selection is extended to the start of the document.

```
PageUp 1, 1
If CmpBookmarks("\StartOfSel", "\StartOfDoc") = 0 Then
    Print "Selected to start of document."
End If
```

PageUp, PageUp()

¬Example

PageUp [*Count*] [**,** *Select*]

PageUp([*Count*] [**,** *Select*]**)**

The **PageUp** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+PAGE UP) up by the specified number of screens, where one screen is equal to the height of the active window. If there is not a full screen between the insertion point or the selection and the beginning of the document, **PageUp** moves the insertion point or the active end of the selection to the beginning of the document.

Argument	Explanation
Count	The number of screens to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection down.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **PageUp** moves the insertion point *Count*-1 screens above the selection. Nonzero Text is selected. If there is already a selection, **PageUp** moves the active end of the selection toward the beginning of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **PageUp** shrinks the selection. In a selection made from right to left, it extends the selection.

The **PageUp()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved up.
-1	If the insertion point or the active end of the selection is moved up by any number of screens, even if less than *Count*. For example, PageUp(2) returns -1 even if a single **PageUp** instruction brings the beginning of the document into view.

See also

Moving the Insertion Point and Selecting Statements and Functions

EditGoTo

HPage

HScroll

PageDown

PrevPage

VPage
VScroll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ParaDown Example

This example inserts three asterisks (***) before each paragraph in the document. Because the While ParaDown() instruction eventually moves the insertion point to the end of the last paragraph, where the asterisks are not appropriate, the **If** control structure uses **CmpBookmarks()** to detect that condition and exit the **While...Wend** loop.

```
StartOfDocument
Insert "***"
While ParaDown()
    If CmpBookmarks("\Sel", "\EndOfDoc") <> 0 Then
        Insert "***"
    End If
Wend
```

ParaDown, ParaDown()

¬Example

ParaDown [*Count*] [**,** *Select*]

ParaDown([*Count*] [**,** *Select*]**)**

The **ParaDown** statement moves the insertion point or the active end of the selection (the end that moves when you press CTRL+SHIFT+DOWN ARROW) down by the specified number of paragraphs.

Regardless of the position of the insertion point within a paragraph or whether there is a selection within a paragraph, the instruction ParaDown moves the insertion point to the start of the next paragraph. The only exception is when the insertion point is in the last paragraph of the document; in that case, ParaDown moves the insertion point to the end of the paragraph. If the current selection extends over multiple paragraphs, the instruction ParaDown moves the insertion point to the beginning of the first paragraph following the end of the selection.

Argument	Explanation
Count	The number of paragraphs to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection up.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **ParaDown** moves the insertion point *Count* paragraphs below the selection. Nonzero Text is selected. If there is already a selection, **ParaDown** moves the active end of the selection toward the end of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **ParaDown** extends the selection. In a selection made from right to left, it shrinks the selection.

The **ParaDown()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved down.
-1	If the insertion point or the active end of the selection is moved down by any number of paragraphs, even if less than *Count*. For example, ParaDown(2) returns -1 even if the insertion point is at the beginning of the last paragraph in the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

LineDown

PageDown
ParaUp

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ParaUp Example

This example finds the next occurrence of the phrase "Press F1 for Help" and then selects the entire text of the surrounding paragraph:

```
EditFind .Find = "Press F1 for Help"
ParaUp
ParaDown 1, 1
```

Note that you can also use the predefined bookmark "\Para" in the instruction EditGoTo "\Para" to select the paragraph containing the insertion point or selection.

ParaUp, ParaUp()

¬Example

ParaUp [*Count*] [**,** *Select*]

ParaUp([*Count*] [**,** *Select*]**)**

The **ParaUp** statement moves the insertion point or the active end of the selection (the end that moves when you press CTRL+SHIFT+UP ARROW) up by the specified number of paragraphs.

If the insertion point is at the beginning of a paragraph, the instruction ParaUp moves the insertion point to the beginning of the previous paragraph. If the insertion point is not at the beginning of the paragraph, or if there is a selection, ParaUp moves the insertion point to the beginning of the current paragraph. If the current selection extends over multiple paragraphs, the instruction ParaUp moves the insertion point to the beginning of the first selected paragraph.

Argument	Explanation
Count	The number of paragraphs to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection down.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **ParaUp** moves the insertion point *Count*-1 paragraphs above the selection. Nonzero Text is selected. If there is already a selection, **ParaUp** moves the active end of the selection toward the beginning of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **ParaUp** shrinks the selection. In a selection made from right to left, it extends the selection.

The **ParaUp()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved up.
-1	If the insertion point or the active end of the selection is moved up by any number of paragraphs, even if less than *Count*. For example, ParaUp(2) returns -1 even if the insertion point is at the end of the first paragraph in the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

LineUp

PageUp
ParaDown

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PrevCell Example

If the insertion point is in the first cell, this example selects the entire table:

```
If PrevCell() = 0 Then TableSelectTable
```

PrevCell, PrevCell()

¬Example

PrevCell

PrevCell()

The **PrevCell** statement selects the contents of the previous cell (the same as pressing SHIFT+TAB in a table). If there is already a selection including multiple cells, **PrevCell** selects the first cell in the selection.

The **PrevCell()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the selection is wholly contained in the leftmost cell of the first row (in other words, if the selection is not moved)
-1	If the selection is moved

See also

Moving the Insertion Point and Selecting Statements and Functions

NextCell

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PrevField Example

This macro counts the number of fields above the insertion point (excluding XE, TA, TC, and RD fields) and displays the result in a message box:

```
Sub MAIN
    count = 0
    While PrevField()
        count = count + 1
    Wend
    MsgBox "Fields above the insertion point:" + Str$(count)
End Sub
```

PrevField, PrevField()

¬Example

PrevField

PrevField()

The **PrevField** statement selects the previous field, regardless of whether the field is showing its codes or result. **PrevField** skips over the following fields, which are formatted as hidden text: XE (Index Entry), TA (Table of Authorities Entry), TC (Table of Contents Entry), and RD (Referenced Document).

The **PrevField()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If there is no previous field (in other words, if the selection is not moved)
1	If the selection is moved

With field codes displayed, you can use **EditFind** to go to the previous field of any type, including XE, TA, TC, and RD fields, provided that hidden text is displayed. Simply specify "^d" (the code for a field character) as the text for the .Find argument and 1 for the .Direction argument, as shown in the following instructions:

```
ViewFieldCodes 1
EditFind .Find = "^d", .Direction = 1, .Format = 1
```

To find only XE fields, specify "^d XE" as the text to find.

See also

Moving the Insertion Point and Selecting Statements and Functions

NextField

PrevObject

PrevObject

Moves the insertion point to the previous document object on the current page in page layout view (the same as pressing ALT+UP ARROW). Document objects include frames, footnotes, table cells, and text columns.

From the top of the first text column on a page, the **PrevObject** statement moves through document objects in the following order:

- Frames, if any appear, in the order that they appear in normal view, from the following page break to the previous page break
- Each text column, in order from last to first (if there is more than one), moving through all footnotes and then all table cells that appear within the text column

PrevObject finally moves to the top of the first text column before cycling through the document objects again.

Keep the following points in mind when using **PrevObject**:

- You can identify document objects by selecting the Text Boundaries check box on the View tab in the Options dialog box (Tools menu).
- Word does not consider OLE objects, such as embedded drawings and charts, to be document objects unless they are framed.
- If the insertion point is in a header or footer, **PrevObject** moves between the header and footer.
- Make sure to switch to page layout view before using **PrevObject**.

See also

Moving the Insertion Point and Selecting Statements and Functions

NextObject

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PrevPage Example

This example ensures the document is in page layout view, scrolls to the previous page, and moves the insertion point to the top of the window:

```
ViewPage
PrevPage
StartOfWindow
```

PrevPage, PrevPage()

¬Example

PrevPage

PrevPage()

The **PrevPage** statement scrolls back one page in page layout view without moving the insertion point (the same as clicking the Page Back button at the bottom of the vertical scroll bar in page layout view). If you want to move the insertion point after scrolling, include a **StartOfWindow** statement following **PrevPage** in your macro.

PrevPage scrolls from the current location on one page to the same relative location on the previous page. To scroll to the beginning of the previous page regardless of what portion of the current page appears, use the instruction EditGoTo .Destination = "p-1" instead of **PrevPage**. Note, however, that **EditGoTo** also moves the insertion point.

The **PrevPage()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If there is no previous page (in other words, if the document doesn't scroll)
1	If the document scrolls

Note

PrevPage and **PrevPage()** are available only in page layout view and generate an error if run in another view.

See also

Moving the Insertion Point and Selecting Statements and Functions

EditGoTo

NextPage

PageUp

ViewPage

VPage

PrevWindow

PrevWindow

Activates the window listed immediately before the active window on the Window menu. If the active window is first on the list, **PrevWindow** activates the window at the bottom of the list. This statement does nothing if the active window is the only open window.

Note

Window names are listed on the Window menu in alphabetic order. A path is included in a window name if the file is stored in a directory other than the current directory. When you change the current directory --- for example, by selecting a new directory in the Open dialog box (File menu) --- the window names may change, causing Word to re-sort them alphabetically.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

Activate

ChDir

NextWindow

Window()

WindowList

WindowName$()

Window*Number*

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurAlignment Example

This example positions the insertion point at the beginning of the first subsequent paragraph that does not have the same alignment as the current paragraph. If the alignment is the same to the end of the document, Word displays a message box.

```
SelectCurAlignment
CharRight
If CmpBookmarks("\Sel", "\EndOfDoc") = 0 Then
    MsgBox "No variation in alignment found."
End If
```

SelectCurAlignment

¬Example

SelectCurAlignment

Extends the selection forward until text with a different paragraph alignment is encountered. There are four types of paragraph alignment: left, centered, right, and justified.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

CenterPara

FormatParagraph

JustifyPara

LeftPara

RightPara

SelectCurIndent

SelectCurSpacing

SelectCurTabs

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurColor Example

This example extends the selection from the beginning of the document to the first character formatted with a different color and then displays the number of characters in the selection:

```
StartOfDocument
SelectCurColor
n = Len(Selection$())
MsgBox "Contiguous characters with the same color:" + Str$(n)
```

SelectCurColor

¬Example

SelectCurColor

Extends the selection forward until text with a different color is encountered.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

CharColor

FormatFont

SelectCurFont

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurFont Example

This example extends the selection to the first character in a different font or font size and then increases the font size to the next available setting:

```
SelectCurFont
GrowFont
```

SelectCurFont

¬Example

SelectCurFont

Extends the selection forward until text in a different font or font size is encountered.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

Font

FontSize

FormatFont

SelectCurColor

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurIndent Example

This example determines whether all the paragraphs in the document are formatted with the same left and right indents and then displays a message box indicating the result:

```
StartOfDocument
SelectCurIndent
LineDown
If LineDown() = 0 Then
    MsgBox "All paragraphs share the same left and right indents."
Else
    MsgBox "Not all paragraphs share the same left and right indents."
End If
```

SelectCurIndent

¬Example

SelectCurIndent

Extends the selection forward from the insertion point until text with different left or right paragraph indents is encountered.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

FormatParagraph

Indent

SelectCurAlignment

SelectCurSpacing

SelectCurTabs

UnIndent

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurSentence Example

This example defines the variable a$ as the text in the current sentence. To avoid an error if there is a selection that extends over multiple sentences, the first instruction cancels the selection if there is one.

```
If SelType() = 2 Then SelType 1                    'Cancel selection, if any
SelectCurSentence
a$ = Selection$()
```

SelectCurSentence

¬Example

SelectCurSentence

Selects the entire sentence containing the insertion point or selection, including the trailing space. If the selection is larger than a sentence when **SelectCurSentence** is run, an error occurs.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

SelectCurWord

SentLeft

SentRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurSpacing Example

This example demonstrates a quick way to jump to the beginning of the next paragraph whose line spacing is different from that of the current paragraph. If the spacing is the same to the end of the document, Word displays a message box.

```
SelectCurSpacing
CharRight
If CmpBookmarks("\Sel", "\EndOfDoc") = 0 Then
    MsgBox "No variation in spacing found."
End If
```

SelectCurSpacing

¬Example

SelectCurSpacing

Extends the selection from the insertion point forward until a paragraph with different line spacing is encountered.

See also

Moving the Insertion Point and Selecting Statements and Functions

FormatParagraph

SelectCurAlignment

SelectCurIndent

SelectCurTabs

SpacePara1

SpacePara15

SpacePara2

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurTabs Example

This example determines the position of the first tab stop in the current paragraph, and then sets the same tab stop for the next series of paragraphs that have tab settings different from those in the original paragraph. Because **NextTab()** returns a position in points, points are also used in the **FormatTabs** instruction.

```
n = NextTab(0)
SelectCurTabs
CharRight
SelectCurTabs
FormatTabs .Position = Str$(n) + "pt"
```

SelectCurTabs

¬Example

SelectCurTabs

Extends the selection forward from the insertion point until a paragraph with different tab stops is encountered.

See also

Moving the Insertion Point and Selecting Statements and Functions

FormatTabs

SelectCurAlignment

SelectCurIndent

SelectCurSpacing

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SelectCurWord Example

This example defines the variable a$ as the characters in the word containing the insertion point. If there is a selection, the first instruction cancels it, so in effect, a$ is defined as the first word in the selection.

```
If SelType() = 2 Then SelType 1                    'Cancel selection, if any
SelectCurWord
a$ = Selection$()
```

SelectCurWord

¬Example

SelectCurWord

Selects the entire word containing the insertion point or selection. Whereas double-clicking a word selects both the word and the trailing space (if any), **SelectCurWord** does not select the trailing space. If the selection is larger than a word when **SelectCurWord** is run, an error occurs.

See also

Moving the Insertion Point and Selecting Statements and Functions

SelectCurSentence

WordLeft

WordRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SentLeft Example

This example deletes all sentences in the document containing the phrase "see page." In other words, the example deletes all cross-references to other pages. Note that you could substitute SelectCurSentence for the pair of instructions SentLeft and SentRight 1, 1.

```
StartOfDocument
EditFind .Find = "see page", .WholeWord = 1, .Direction = 0, .Format = 0
While EditFindFound()
    SentLeft
    SentRight 1, 1
    EditClear
    EditFind .Find = "see page", .WholeWord = 1, .Direction = 0, \
        .Format = 0
Wend
```

SentLeft, SentLeft()

¬Example

SentLeft [*Count***,**] [*Select*]

SentLeft([*Count***,**] [*Select*]**)**

The **SentLeft** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+an arrow key) to the left by the specified number of sentences.

Argument	Explanation
Count	The number of sentences to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection to the right.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **SentLeft** moves the insertion point *Count*-1 sentences to the left of the selection. Nonzero Text is selected. If there is already a selection, **SentLeft** moves the active end of the selection toward the beginning of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **SentLeft** shrinks the selection. In a selection made from right to left, it extends the selection.

Note that Word counts empty table cells as "sentences," and that, regardless of length or punctuation, Word considers every paragraph to contain at least one sentence.

The **SentLeft()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved to the left.
-1	If the insertion point or the active end of the selection is moved to the left by any number of sentences, even if less than *Count*. For example, SentLeft(10) returns -1 even if the insertion point is only three sentences from the start of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

CharLeft

ParaUp

SelectCurSentence

SentRight
StartOfLine
WordLeft

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SentRight Example

This example counts the number of sentences in the document (excluding paragraph marks) and displays the result in a message box:

```
StartOfDocument
count = 0
While SentRight() <> 0
    If Right$(Selection$(), 1) <> Chr$(13) Then count = count + 1
Wend
MsgBox "Number of sentences in document:" + Str$(Count)
```

SentRight, SentRight()

¬Example

SentRight [*Count***,**] [*Select*]

SentRight([*Count***,**] [*Select*]**)**

The **SentRight** statement moves the insertion point or the active end of the selection (the end that moves when you press SHIFT+an arrow key) to the right by the specified number of sentences.

Argument	Explanation
Count	The number of sentences to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection to the left.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **SentRight** moves the insertion point *Count*-1 sentences to the right of the selection. Nonzero Text is selected. If there is already a selection, **SentRight** moves the active end of the selection toward the end of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **SentRight** extends the selection. In a selection made from right to left, it shrinks the selection.

Note that Word counts empty table cells as "sentences," and that, regardless of length or punctuation, Word considers every paragraph to contain at least one sentence.

The **SentRight()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved to the right.
-1	If the insertion point or the active end of the selection is moved to the right by any number of sentences, even if less than *Count*. For example, SentRight(10) returns -1 even if the insertion point is only three sentences from the end of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

CharRight

EndOfLine

ParaDown

SelectCurSentence
SentLeft
WordRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SetSelRange Example

This example selects the first 20 characters in the document:

```
SetSelRange 0, 20
```

SetSelRange

¬Example

SetSelRange *Pos1*, *Pos2*

Selects the characters between character position *Pos1* and character position *Pos2*. The character position at the start of the document is 0 (zero), the position after the first character is 1, the position after the second character is 2, and so on. All characters, including nonprinting characters, are counted. Hidden characters are counted even if they are not displayed. Note that the document does not scroll even if the selected characters are not visible in the document window.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

GetSelEndPos()

GetSelStartPos()

GetText$()

ShrinkSelection

ShrinkSelection

Shrinks the selection to the next smaller unit of text. The progression is as follows: entire document, section, paragraph, sentence, word, insertion point. The insertion point is the beginning of the original selection. Running **ShrinkSelection** when there is no selection produces a beep.

Unlike **ExtendSelection**, **ShrinkSelection** does not activate or require extend mode.

_____

See also

Moving the Insertion Point and Selecting Statements and Functions

ExtendMode()

ExtendSelection

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

StartOfColumn Example

This example moves the insertion point from anywhere in the table to the end of the first row:

```
TableSelectRow
StartOfColumn
```

StartOfColumn, StartOfColumn()

¬Example

StartOfColumn [*Select*]

StartOfColumn([*Select*]**)**

The **StartOfColumn** statement moves the insertion point or extends the selection (if *Select* is nonzero) to the top of the table column containing the insertion point or selection. If the selection extends over more than one column, the insertion point moves to the top of the column containing the end of the selection. If the insertion point or selection is not in a table, an error occurs.

Note

If the first row in the table does not have a cell that corresponds to the column that contains the current selection --- for example, if you have deleted or merged cells in the first row --- **StartOfColumn** moves the insertion point to the end of the first row in the table.

The **StartOfColumn()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point was not moved or the selection was not extended (that is, if it was already at the top of the column)
-1	If the insertion point was moved or the selection was extended

If the insertion point is not in a table, **StartOfColumn** and **StartOfColumn()** generate an error.

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfColumn

StartOfRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

StartOfDocument Example

This example moves the insertion point to the start of the document and then prepares Word to begin a replace operation:

```
StartOfDocument
EditFindClearFormatting
EditReplaceClearFormatting
'Instructions that perform a replace operation go here
```

StartOfDocument, StartOfDocument()

¬Example

StartOfDocument [*Select*]

StartOfDocument([*Select*]**)**

The **StartOfDocument** statement moves the insertion point or, if *Select* is nonzero, the active end of the selection (the end that moves when you press CTRL+SHIFT+HOME) to the beginning of the document.

The **StartOfDocument()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection was not moved (for example, if it was already at the start of the document)
-1	If the insertion point or the active end of the selection was moved

See also

Moving the Insertion Point and Selecting Statements and Functions

AtStartOfDocument()

EndOfDocument

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

StartOfLine Example

This example moves the insertion point to the start of the current paragraph regardless of the length of the paragraph or the position of the insertion point within it. The first instruction selects the entire paragraph.

```
EditGoTo "\Para"
StartOfLine
```

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

StartOfLine, StartOfLine()

¬Example

StartOfLine [*Select*]

StartOfLine ([*Select*]**)**

The **StartOfLine** statement moves the insertion point or, if *Select* is nonzero, the active end of the selection (the end that moves when you press SHIFT+HOME) to the beginning of the current line or the line that contains the active end of the selection.

The **StartOfLine()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection was not moved (that is, if it was already at the beginning of the line)
-1	If the insertion point or the active end of the selection was moved

Avoid using **StartOfLine** by itself to go to the beginning of a paragraph unless you are sure that the paragraph is a single line (for example, a word in a list of words).

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfLine

ParaUp

StartOfRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

StartOfRow Example

This example moves the insertion point from anywhere in the table to the beginning of the first cell:

```
StartOfRow
StartOfColumn
```

StartOfRow, StartOfRow()

¬Example

StartOfRow [*Select*]

StartOfRow ([*Select*]**)**

The **StartOfRow** statement moves the insertion point or extends the selection (if *Select* is nonzero) to the beginning of the first cell in the table row containing the insertion point. If the selection extends over more than one row, the insertion point moves or the selection is extended to the first cell of the first row in the selection. If the insertion point or selection is not in a table, an error occurs.

The **StartOfRow()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point was not moved or the selection was not extended (that is, if it was already at the beginning of the row)
-1	If the insertion point was moved or the selection was extended

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfRow

StartOfColumn

StartOfLine

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

StartOfWindow Example

This example selects the visible contents of the active document window:

```
StartOfWindow
EndOfWindow 1
```

StartOfWindow, StartOfWindow()

¬Example

StartOfWindow [*Select*]

StartOfWindow([*Select*]**)**

The **StartOfWindow** statement moves the insertion point or, if *Select* is nonzero, the active end of the selection (the end that moves when you press CTRL+SHIFT+PAGE UP) to the upper-left corner of the contents currently visible in the document window.

The **StartOfWindow()** function behaves the same as the statement and also returns one of the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection was not moved (that is, if it was already at the upper-left corner of the window)
-1	If the insertion point or the active end of the selection was moved

See also

Moving the Insertion Point and Selecting Statements and Functions

EndOfWindow

StartOfDocument

VLine

VLine [*Count*]

Scrolls the active document vertically. A "line" corresponds to clicking a scroll arrow on the vertical scroll bar once.

Argument	Explanation
Count	The amount to scroll, in lines: Omitted One line down > 0 (zero) The specified number of lines down < 0 (zero) The specified number of lines up

See also

Moving the Insertion Point and Selecting Statements and Functions

HLine

VPage

VScroll

VPage

VPage [*Count*]

Scrolls the active document vertically. **VPage** corresponds to clicking the vertical scroll bar above or below the scroll box.

Argument	Explanation
Count	The amount to scroll, in screens: Omitted One screen down > 0 (zero) The specified number of screens down < 0 (zero) The specified number of screens up

See also

Moving the Insertion Point and Selecting Statements and Functions

HPage

VLine

VScroll

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

VScroll Example

This example scrolls to the middle of the active document:

```
VScroll 50
```

VScroll, VScroll()

¬Example

VScroll *Percentage*

VScroll()

The **VScroll** statement scrolls vertically to the specified percentage of the document length. **VScroll** corresponds to dragging the scroll box on the vertical scroll bar.

The **VScroll()** function returns the current vertical scroll position as a percentage of the document length.

See also

Moving the Insertion Point and Selecting Statements and Functions

HScroll

VLine

VPage

WordLeft, WordLeft()

WordLeft [*Count***,**] [*Select*]

WordLeft([*Count***,**] [*Select*]**)**

The **WordLeft** statement moves the insertion point or the active end of the selection (the end that moves when you press CTRL+SHIFT+LEFT ARROW) to the left by the specified number of words.

Argument	Explanation
Count	The number of words to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection to the right.
Select	Specifies whether to select text: 0 (zero) or omitted — Text is not selected. If there is already a selection, **WordLeft** moves the insertion point *Count*-1 words to the left of the selection. Nonzero — Text is selected. If there is already a selection, **WordLeft** moves the active end of the selection toward the beginning of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **WordLeft** shrinks the selection. In a selection made from right to left, it extends the selection.

Note that Word includes spaces following a word as part of the word. However, Word counts punctuation, tab characters, and paragraph marks as "words." For example, if a word is enclosed in quotation marks, moving the insertion point from the position following the closing quotation mark to the position preceding the opening quotation mark using **WordLeft** would require the instruction WordLeft 3.

The **WordLeft()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved to the left.
-1	If the insertion point or the active end of the selection is moved to the left by any number of words, even if less than *Count*. For example, WordLeft(10) returns -1 even if the insertion point is only three words from the start of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

CharLeft

SelectCurWord

SentLeft
WordRight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

WordRight Example

This example counts the number of words (including punctuation, tab characters, and paragraph marks) in the selection and then displays the result in a message box:

```
EditBookmark "CountMe", .Add
SelType 1
While CmpBookmarks("\Sel", "CountMe") = 6 \
        Or CmpBookmarks("\Sel", "CountMe") = 8
    WordRight
    count = count + 1
Wend
EditGoTo "CountMe"
EditBookmark "CountMe", .Delete
MsgBox "There are" + Str$(count) + " words in the selection."
```

WordRight, WordRight()

¬Example

WordRight [*Count*,] [*Select*]

WordRight([*Count*,] [*Select*]**)**

The **WordRight** statement moves the insertion point or the active end of the selection (the end that moves when you press CTRL+SHIFT+RIGHT ARROW) to the right by the specified number of words.

Argument	Explanation
Count	The number of words to move; if omitted, 1 is assumed. Negative values move the insertion point or the active end of the selection to the left.
Select	Specifies whether to select text: 0 (zero) or omitted Text is not selected. If there is already a selection, **WordRight** moves the insertion point *Count*-1 words to the right of the selection. Nonzero Text is selected. If there is already a selection, **WordRight** moves the active end of the selection toward the end of the document. In a typical selection made from left to right, where the active end of the selection is closer to the end of the document, **WordRight** extends the selection. In a selection made from right to left, it shrinks the selection.

Note that Word includes spaces following a word as part of the word. However, Word counts punctuation, tab characters, and paragraph marks as "words."

The **WordRight()** function behaves the same as the statement and also returns the following values.

Value	Explanation
0 (zero)	If the insertion point or the active end of the selection cannot be moved to the right.
-1	If the insertion point or the active end of the selection is moved to the right by any number of characters, even if less than *Count*. For example, WordRight(10) returns -1 even if the insertion point is only three words from the end of the document.

See also

Moving the Insertion Point and Selecting Statements and Functions

CharRight

SelectCurWord

SentRight

WordLeft

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ActivateObject Example

This example moves the insertion point to the next Microsoft WordArt object and activates it for editing:

```
EditGoTo "o'MSWordArt.2'"
ActivateObject
```

ActivateObject

¬Example

ActivateObject

Activates the selected embedded object for editing, or plays the sound or video file associated with the selected icon (for example, a sound file associated with the microphone icon). **ActivateObject** is equivalent to double-clicking the selected embedded object.

See also

Object Linking and Embedding Statements and Functions

EditGoTo

EditObject

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditLinks Examples

This example unlocks an existing link to a Microsoft Excel 5.0 worksheet and makes updating automatic:

```
EditLinks .UpdateMode = 0, .Locked = 0, .Link = "\Sel", \
    .Application = "Excel.Sheet.5", .Item = "R1C1:R5C3", \
    .FileName = "C:\XLS\SALES.XLS"
```

The following example selects each link between the active document and a Microsoft Excel 5.0 worksheet and makes updating manual:

```
StartOfDocument
ViewFieldCodes 1
EditFind "^d LINK Excel.Sheet.5", .Format = 0, .Wrap = 0
While EditFindFound()
    Dim dlg As EditLinks
    GetCurValues dlg
    dlg.UpdateMode = 1
    EditLinks dlg
    CharRight
    RepeatFind
Wend
```

EditLinks

¬Example

EditLinks [**.UpdateMode** = *number,*] [**.Locked** = *number,*] [**.SavePictureInDoc =** *number,*] [**.UpdateNow,**] [**.OpenSource,**] [**.KillLink,**] **.Link** = *text,* **.Application** = *text,* **.Item** = *text,* **.FileName** = *text*

Sets options for the specified link. The arguments for the **Link** statement correspond to the options in the Links dialog box (Edit menu).

Argument	Explanation
.UpdateMode	Specifies how the link is updated: 0 (zero) Automatically 1 Manually
.Locked	Specifies whether to lock or unlock the link: 0 (zero) Unlock the link. 1 Lock the link.
.SavePictureInDoc	If 1, saves a copy of the linked object in the Word document.
.UpdateNow	Updates the specified link.
.OpenSource	Opens the source of the specified link (for example, a Microsoft Excel worksheet).
.KillLink	Replaces the specified link with its most recent result.
.Link	Specifies the link listed in the Links dialog box whose options are to be set: "1" corresponds to the first link in the list, "2" corresponds to the second, and so on. Note that the number must be enclosed in quotation marks. You can also use the "\Sel" predefined bookmark to specify the currently selected link.
.Application	The type of document supplying the link, as it appears in the LINK field codes---for example, "Word.Document.6" for a Word document or "Excel.Sheet.5" for a Microsoft Excel 5.0 worksheet.
.Item	Text that identifies the linked information (for example, a bookmark name such as "INTERN_LINK2" in a Word document, or a range of cells such as R1C1:R5C3 in a Microsoft Excel worksheet).
.FileName	The path and filename of the source document containing the linked item.

The last three arguments correspond to choosing the Change Source button and setting options in the Change

Source dialog box.

See also

Object Linking and Embedding Statements and Functions

EditPasteSpecial

InsertField

LockFields

UnlinkFields

UnlockFields

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditObject Example

This example selects the next equation in the active document and opens it for editing in the Equation Editor:

```
EditGoTo "q"
EditObject
```

EditObject

¬Example

EditObject

Opens the selected object linking and embedding (OLE) object for editing in the application associated with the object. OLE objects include Equation Editor equations, font effects created with WordArt, Microsoft Excel charts, and so on.

_____

See also

Object Linking and Embedding Statements and Functions

ActivateObject

EditGoTo

InsertObject

EditPasteSpecial

EditPasteSpecial [**.IconNumber** = *number*] [**, .Link** = *number*] [**, .DisplayIcon =** *number*] [**, .Class** = *text*] [**, .DataType** = *text*] [**, .IconFilename** = *text*] [**, .Caption** = *text*]

Inserts the contents of the Clipboard at the insertion point. The arguments for the **EditPasteSpecial** statement correspond to the options in the Paste Special dialog box (Edit menu). Unlike **EditPaste**, **EditPasteSpecial** allows you to control the format of the pasted information and to establish a link with the information source (for example, a Microsoft Excel worksheet).

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.
.Link	Specifies whether or not to create a link: 0 (zero) or omitted No link is created. 1 A link is created.
.DisplayIcon	Specifies whether or not to display a link as an icon: 0 (zero) or omitted Link is not displayed as an icon. 1 Link is displayed as an icon.
.Class	The object class of the Clipboard contents. This argument cannot be set. However, you can define a dialog record as **EditPasteSpecial** and then use the dialog record to return the current value of .Class.
.DataType	Specifies a format for the pasted contents: Bitmap Windows bitmap DIB Windows Device Independent Bitmap Object Object linking and embedding (OLE) object PICT Windows metafile RTF Rich-text format Text Unformatted text
.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.

Note

Not all data types are available for each Clipboard item. For example, if the Clipboard contains a text selection from a Word document, the available data types are "Object," "PICT, "RTF," and "Text." For a selection from a Microsoft Excel worksheet, the available formats are "Bitmap," "Object," "PICT," "RTF," and "Text." If you specify an unavailable data type, an error occurs.

For more information on linking, see Exchanging Information with Other Applications.

See also

Object Linking and Embedding Statements and Functions

EditCopy

EditCut

EditLinks

EditPaste

InsertField

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

EditPicture Example

This example goes to the next graphic and opens it for editing.

```
EditGoTo "g"
CharRight 1, 1
EditPicture
```

EditPicture

¬Example

EditPicture

Opens the selected graphic for editing.

_____

See also

Object Linking and Embedding Statements and Functions

EditObject

FormatPicture

InsertPicture

Picture

InsertChart

InsertChart

Starts Microsoft Graph and displays a default chart for editing. When the user exits Microsoft Graph, the chart is embedded in the active document. The instruction InsertChart is the same as the instruction InsertObject .Class = "MSGraph".

_____

See also

Object Linking and Embedding Statements and Functions

InsertDrawing

InsertExcelTable

InsertObject

InsertDatabase

InsertDatabase [**.Format** = *number*] [**, .Style** = *number*] [**, .LinkToSource** = *number*] [**, .Connection** = *text*] [**, .SQLStatement** = *text*] [**, .SQLStatement1** = *text*] [**, .PasswordDoc** = *text*] [**, .PasswordDot** = *text*] [**, .DataSource** = *text*] [**, .From** = *text*] [**, .To** = *text*] [**, .IncludeFields** = *number*]

Retrieves data from a data source (for example, a separate Word document, a Microsoft Excel worksheet, or a Microsoft Access database) and inserts it into the active document as a table.

Argument	Explanation
.Format	Specifies a format listed under Formats in the Table AutoFormat dialog box (Table menu): 0 (zero) corresponds to the first format in the list ("none"), 1 corresponds to the second format, and so on.
.Style	Specifies which attributes of the format specified by .Format to apply to the table. Use the sum of any combination of the following values: 0 None 1 Borders 2 Shading 4 Font 8 Color 16 Best Fit 32 Heading Rows 64 Last Row 128 First Column 256 Last Column
.LinkToSource	If 1, establishes a link between the new table and the data source.
.Connection	A connection string used by an ODBC driver. For information on valid connection strings, see the Help file for the appropriate ODBC driver.
.SQLStatement	An optional query string that retrieves a subset of the data in a primary data source to insert into the document. For information on valid query strings, see the Help file for the appropriate ODBC driver.
.SQLStatement 1	If the query string is longer than 255 characters, .SQLStatement specifies the first portion of the string and .SQLStatement1 specifies the second portion.
.PasswordDoc	The password (if any) required to open the data source.
.PasswordDot	If the data source is a Word

	document, the password (if any) required to open the attached template.
.DataSource	The path and filename of the data source.
.From	The number of the first data record in the range of records you want to insert.
.To	The number of the last data record in the range of records you want to insert.
.IncludeFields	If 1, field names from the data source appear in the first row of the new table.

Note

Help files for ODBC drivers are located in the SYSTEM subdirectory of your Windows directory. Filenames begin with the letters "DRV" and end with the .HLP filename extension (for example, DRVFOX.HLP and DRVACCSS.HLP).

See also

Object Linking and Embedding Statements and Functions

InsertExcelTable

MailMergeCreateDataSource

InsertEquation

InsertEquation

If Microsoft Equation Editor is installed on your computer, starts Equation Editor. When the user exits Equation Editor, the equation is embedded in the active document. The instruction InsertEquation is the same as the instruction InsertObject .Class = "Equation".

_____

See also

Object Linking and Embedding Statements and Functions

ActivateObject

EditObject

InsertChart

InsertDrawing

InsertExcelTable

InsertWordArt

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertObject Examples

This example starts Microsoft Paintbrush. When the user chooses Exit & Return from the File menu, Word inserts the following field at the insertion point: **{**EMBED PBrush * MERGEFORMAT**}.**

```
InsertObject .Class = "PBrush"
```

The following example inserts an embedded object that is linked to the file FLOCK.BMP:

```
InsertObject .FileName = "C:\WIN31\FLOCK.BMP", .Link = 1, \
        .Tab = 1, .Class = "PBrush"
```

InsertObject

¬Example

InsertObject [**.IconNumber** = *number*] [**, .FileName** = *text*] [**, .Link** = *number*] [**, .DisplayIcon** = *number*] [**, .Tab** = *number*] [**, .Class** = *text*] [**, .IconFilename** = *number*] [**, .Caption** = *text*]

Starts an object linking and embedding (OLE) application in which the user creates an object, or immediately creates an embedded object using a specified file. In either case, **InsertObject** inserts an EMBED field at the insertion point.

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.
.FileName	The path and filename of the file you want to store as an embedded object in the active document. If you specify .Filename, you must set .Tab to 1 and specify .Class. For an example of inserting an existing file as an object, see the second example in this entry.
.Link	If 1, links the embedded object to the file specified by .FileName. When the file changes, Word updates the embedded object.
.DisplayIcon	Specifies whether or not to display a link as an icon: 0 (zero) or omitted Link is not displayed as an icon. 1 Link is displayed as an icon.
.Tab	Specifies which tab to select when you display the Object dialog box with a **Dialog** or **Dialog()** instruction. 0 (zero) Create New tab 1 Create From File tab
.Class	The class name of a new object to insert, as listed in the Create New tab of the Object dialog box (Insert menu)
.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.

See also

Object Linking and Embedding Statements and Functions

ActivateObject

EditObject

InsertChart

InsertDrawing

InsertExcelTable

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertPicture Example

This example inserts an INCLUDEPICTURE field whose result is the bitmap WINLOGO.BMP:

```
InsertPicture .Name = "WINLOGO.BMP", .LinkToFile = 1
```

InsertPicture

¬Example

InsertPicture .Name = *text* [**, .LinkToFile =** *number*] [**, .New**]

Inserts a graphic at the insertion point. If .LinkToFile is set to 1 or 2, the graphic is inserted as an INCLUDEPICTURE field that you can update when the graphics file changes.

Argument	Explanation
.Name	The path and filename of the graphic to insert. If the path is omitted, the current directory is assumed.
.LinkToFile	Specifies whether to insert the graphic as a field and whether to save graphic data in the document: 0 (zero) or omitted — Inserts the graphic specified in the .Name argument. 1 — Inserts an INCLUDEPICTURE field at the insertion point and saves graphic data in the document. 2 — Inserts an INCLUDEPICTURE field at the insertion point and prevents graphic data from being stored in the document by adding a \d switch.
.New	Inserts a 1-inch-square empty metafile graphic, surrounded by a border.

See also

Object Linking and Embedding Statements and Functions

InsertDrawing

InsertFile

InsertObject

InsertSound

InsertSound

Starts Sound Recorder so the user can record or insert a sound. When the user exits Sound Recorder, the sound is embedded in the active document. If no sound equipment is installed on the computer, an error occurs.

See also

Object Linking and Embedding Statements and Functions

InsertChart

InsertDrawing

InsertExcelTable

InsertObject

InsertWordArt

InsertWordArt

If Microsoft WordArt is installed on your computer, starts WordArt. When the user exits WordArt, the WordArt object is embedded in the active document. The instruction InsertWordArt is the same as the instruction InsertObject .Class = "WordArt".

_____

See also

Object Linking and Embedding Statements and Functions

ActivateObject

EditObject

InsertChart

InsertDrawing

InsertEquation

InsertExcelTable

CreateSubdocument

CreateSubdocument

Converts the selected outline headings into subdocuments. If the active document is not in master document or outline view, or if the selection spans a subdocument boundary, an error occurs. An error is also generated if the first paragraph selected is not a heading.

See also

Outlining and Master Documents Statements and Functions

InsertSubdocument

MergeSubdocument

OpenSubdocument

RemoveSubdocument

SplitSubdocument

ViewMasterDocument

DemoteToBodyText

DemoteToBodyText

Demotes the selected headings to body text by applying the Normal style. **DemoteToBodyText** has no effect if the selected paragraphs are already formatted with Normal style.

See also

Outlining and Master Documents Statements and Functions

OutlineDemote

OutlineMoveDown

OutlinePromote

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InsertSubdocument Example

This example adds the document SUB1.DOC to the beginning of the active document as a read-only, or locked, subdocument:

```
StartOfDocument
ViewMasterDocument
InsertSubdocument .Name = "C:\MYDOCS\SUB1.DOC", .ReadOnly = 1
```

InsertSubdocument

¬Example

InsertSubdocument .Name = *text* [**, .ConfirmConversions =** *number*] [**, .ReadOnly =** *number*] [**, .PasswordDoc =** *text*] [**, .PasswordDot =** *text*] [**, .Revert =** *number*] [**, .WritePasswordDoc =** *text*] [**, .WritePasswordDot =** *text*]

Adds the specified file as a subdocument in the active master document. If the active document is not in outline or master document view, an error occurs. For argument descriptions, see FileOpen.

See also

Outlining and Master Documents Statements and Functions

CreateSubdocument

FileOpen

MergeSubdocument

OpenSubdocument

RemoveSubdocument

SplitSubdocument

ViewMasterDocument

MergeSubdocument

MergeSubdocument

Merges the selected subdocuments of a master document into one subdocument. If only one subdocument is selected, or if the active document is not in master document or outline view, an error occurs.

_____

See also

Outlining and Master Documents Statements and Functions

CreateSubdocument

InsertSubdocument

OpenSubdocument

RemoveSubdocument

SplitSubdocument

ViewMasterDocument

OpenSubdocument

OpenSubdocument

Opens the subdocument identified by the location of the insertion point or the beginning of the selection in a master document. Word opens the subdocument in a separate document window. If the active document is not a master document, or if the master document is not in master document or outline view, an error occurs.

_____

See also

Outlining and Master Documents Statements and Functions

CreateSubdocument

FileOpen

InsertSubdocument

MergeSubdocument

RemoveSubdocument

SplitSubdocument

ViewMasterDocument

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OutlineCollapse Example

This example collapses every heading in the document by one level:

ViewOutline

EditSelectAll

OutlineCollapse

OutlineCollapse

¬Example

OutlineCollapse

Collapses one level of heading or body text under the selected headings. If the active document is not in outline or master document view, an error occurs.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineExpand

ShowAllHeadings

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OutlineDemote Example

This example demotes each paragraph in the document that follows a Heading 2:

```
StartOfDocument : ViewOutline
EditFindClearFormatting
EditFindStyle .Style = "Heading 2"
EditFind .Find = "", .Direction = 0, .Format = 1, .Wrap = 0
While EditFindFound()
    ParaDown
    OutlineDemote
    EditFind .Direction = 0
Wend
```

OutlineDemote

¬Example

OutlineDemote

Applies the next heading level style to the selected headings (Heading 1 through Heading 8) or body text.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineMoveDown

OutlinePromote

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OutlineExpand Example

This example expands every heading in the document by one level:

ViewOutline

EditSelectAll

OutlineExpand

OutlineExpand

¬Example

OutlineExpand

Expands one level of heading or body text under the selected headings. If the active document is not in outline or master document view, an error occurs.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineCollapse

ShowAllHeadings

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OutlineLevel() Example

This example deletes all paragraphs in a document that are not formatted with a built-in heading level style:

```
StartOfDocument
ViewNormal
While ParaDown()
    ParaUp                      'Go to each paragraph in turn.
    If OutlineLevel() = 0 Then                  'If it's not a heading,
        ParaDown 1, 1           'select it, then
        EditClear               'delete it.
    Else                        'If it is a heading,
        ParaDown                'go to the next paragraph
    End If
Wend                            'and begin the process again.
```

OutlineLevel()

¬Example

OutlineLevel()

Returns a number corresponding to the heading level of the selected paragraph. Returns 0 (zero) if the style of the selected paragraph is not a built-in heading level style. If multiple paragraphs are selected, **OutlineLevel()** returns the heading level of the first paragraph in the selection.

See also

Outlining and Master Documents Statements and Functions

FormatStyle

StyleName$()

OutlineMoveDown

OutlineMoveDown

Moves the selected paragraphs below the next visible paragraph. Body text moves with a heading only if the body text is selected or collapsed.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineDemote

OutlineMoveUp

OutlineMoveUp

OutlineMoveUp

Moves the selected paragraphs above the next visible paragraph. Body text moves with a heading only if the body text is selected or collapsed.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineMoveDown

OutlinePromote

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OutlinePromote Example

This example promotes the heading level style of the current paragraph if each character in the paragraph, including the paragraph mark, is bold:

```
ViewOutline
EditGoTo "\Para"
If Bold() = 1 Then OutlinePromote
```

OutlinePromote

¬Example

OutlinePromote

Applies the previous heading level style to the selected headings (Heading 2 through Heading 9) or body text.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineDemote

OutlineMoveUp

OutlineShowFirstLine, OutlineShowFirstLine()

OutlineShowFirstLine [*On*]

OutlineShowFirstLine()

The **OutlineShowFirstLine** statement controls the display of body text in outline view. You can use this statement to make the document easier to scan by hiding all but the first line of body text. If the active document is not in outline or master document view, an error occurs.

Argument	Explanation
On	Specifies whether or not to display the first line of body text: 0 (zero) All body text is shown. 1 or omitted Only the first line of body text is shown.

The **OutlineShowFirstLine()** function returns the following values.

Value	Explanation
0 (zero)	All lines of body text are displayed.
-1	Only the first line of body text is displayed.

See also

Outlining and Master Documents Statements and Functions

OutlineCollapse

OutlineShowFormat

OutlineShowFormat

OutlineShowFormat

Displays character formatting in outline view or, if character formatting is already displayed, hides it. If the active document is not in outline or master document view, an error occurs.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineShowFirstLine

ViewDraft

RemoveSubdocument

RemoveSubdocument

Merges the contents of the selected subdocuments into the master document and removes the subdocuments. **RemoveSubdocument** does not delete subdocument files. If the active document is not in master document or outline view, or the insertion point is not in a subdocument, an error occurs.

_____

See also

Outlining and Master Documents Statements and Functions

CreateSubdocument

InsertSubdocument

MergeSubdocument

OpenSubdocument

SplitSubdocument

ViewMasterDocument

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ShowAllHeadings Example

This example switches to outline view and displays all text:

```
ViewOutline
ShowHeading9
ShowAllHeadings
```

ShowAllHeadings

¬Example

ShowAllHeadings

In outline view, toggles between showing all text (headings and body text) and showing only headings. To be sure to display all text, include a **ShowHeading9** instruction to display all headings without body text before running a **ShowAllHeadings** instruction.

ShowAllHeadings is available only in outline view and master document view; an error occurs if the statement is run in another view.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineCollapse

OutlineExpand

OutlineShowFirstLine

ShowHeading*Number*

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ShowHeading*Number* Example

This example takes advantage of outline view to easily reorder sections of the document. Consider an alphabetic reference whose headings, formatted as Heading 1 paragraphs, have gotten out of order; this macro can quickly alphabetize the sections. Collapsed headings and body text move with the Heading 1 paragraphs.

```
ViewOutline
ShowHeading1
EditSelectAll
TableSort .Order = 0
```

ShowHeading*Number*

¬Example

ShowHeading*Number*

In outline view, shows all headings up to the specified heading level and hides subordinate headings and body text. *Number* is an integer from 1 to 9; you cannot use a variable in place of *Number.*

ShowHeading*Number* is available only in outline view and master document view; an error occurs if the statement is run in another view.

_____

See also

Outlining and Master Documents Statements and Functions

OutlineCollapse

OutlineExpand

OutlineShowFirstLine

ShowAllHeadings

SplitSubdocument

SplitSubdocument

Divides an existing subdocument into two subdocuments at the same level in master document or outline view. The division is made at the insertion point (or the beginning of the selection, if there is one). If the active document is not in master document or outline view, or the insertion point is not at the beginning of a paragraph within a subdocument, an error occurs.

_____

See also

Outlining and Master Documents Statements and Functions

CreateSubdocument

InsertSubdocument

MergeSubdocument

OpenSubdocument

RemoveSubdocument

ViewMasterDocument

ViewOutline, ViewOutline()

ViewOutline

ViewOutline()

The **ViewOutline** statement switches the active document to outline view. The **ViewOutline()** function returns the following values.

Value	Explanation
0 (zero)	If the active document is not in outline view
-1	If the active document is in outline view

See also

Outlining and Master Documents Statements and Functions

FilePrintPreview

ViewDraft

ViewMasterDocument

ViewNormal

ViewPage

CenterPara, CenterPara()

CenterPara

CenterPara()

Centers the selected paragraphs.

The **CenterPara()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is centered
-1	If part of the selection is centered or there is a mix of alignments
1	If all the selection is centered

See also

Paragraph Formatting Statements and Functions

FormatParagraph

JustifyPara

LeftPara

RightPara

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CloseUpPara Example

This macro toggles Before formatting between one line and no lines before the selected paragraphs:

```
Sub MAIN
    Dim dlg As FormatParagraph
    GetCurValues dlg
    If Val(dlg.Before) <> 0 Then
        CloseUpPara
    Else
        OpenUpPara
    End If
End Sub
```

CloseUpPara

¬Example

CloseUpPara

Removes the paragraph formatting that creates space before the selected paragraphs. **CloseUpPara** corresponds to setting to 0 (zero) the Before option on the Indents And Spacing tab in the Paragraph dialog box (Format menu).

_____

See also

Paragraph Formatting Statements and Functions

FormatParagraph

OpenUpPara

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatDropCap Example

This example makes the first character of every section a dropped capital letter. The instruction .EditGoTo .Destination = "s" moves the insertion point to the beginning of the following section. As soon as the function CmpBookmarks("tmp", "\Sel") returns a value of 0 (zero), indicating that the insertion point did not move and that there are no more sections, the temporary bookmark is deleted and the macro ends.

```
StartOfDocument
FormatDropCap .Position = 1, .DropHeight = "3"
EditBookmark .Name = "temp", .Add
EditGoTo .Destination = "s"
While CmpBookmarks("temp", "\Sel") <> 0
    FormatDropCap .Position = 1, .DropHeight = "3"
    CharLeft
    EditBookmark .Name = "temp", .Add
    EditGoTo .Destination = "s"
Wend
EditBookmark .Name = "temp", .Delete
```

FormatDropCap

¬Example

FormatDropCap [**.Position =** *number*] [**, .Font =** *text*] [**, .DropHeight =** *number or text*] [**, .DistFromText =** *number or text*]

Inserts a frame for the first character of the current paragraph so that it becomes a dropped capital letter. The arguments for the **FormatDropCap** statement correspond to the options in the Drop Cap dialog box (Format menu).

Argument	Explanation
.Position	Specifies the positioning for the dropped capital letter or removes dropped capital letter formatting 0 (zero) or omitted — No dropped capital letter formatting 1 — Dropped (lines in the paragraph wrap around the dropped capital letter) 2 — In Margin (lines in the paragraph remain flush)
.Font	The font you want for the dropped capital letter
.DropHeight	The height of the dropped capital letter specified in lines of surrounding text
.DistFromText	The distance between the dropped capital letter and the rest of the paragraph in points or a text measurement

See also

Paragraph Formatting Statements and Functions

FormatFrame

InsertFrame

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatParagraph Example

This example sets justified alignment and adds 1 inch of space above and below each paragraph in the selection:

```
FormatParagraph .Alignment = 3, .Before = "1 in", .After = "1 in"
```

FormatParagraph

¬Example

FormatParagraph [**.LeftIndent** = *number or text*] [**, .RightIndent** = *number or text*] [**, .Before** = *number or text*] [**, .After** = *number or text*] [**, .LineSpacingRule** = *number*] [**, .LineSpacing** = *number or text*] [**, .Alignment** = *number*] [**, .WidowControl** = *number*] [**, .KeepWithNext** = *number*] [**, .KeepTogether** = *number*] [**, .PageBreak** = *number*] [**, .NoLineNum** = *number*] [**, .DontHyphen** = *number*] [**, .Tab** = *number*] [**, .FirstIndent** = *number or text*]

Applies paragraph formatting to the selected paragraphs. The arguments for the **FormatParagraph** statement correspond to the options in the Paragraph dialog box (Format menu).

Argument	Explanation
.LeftIndent	The left indent in points or a text measurement.
.RightIndent	The right indent in points or a text measurement.
.Before	The space before the paragraph in points or a text measurement.
.After	The space after the paragraph in points or a text measurement.
.LineSpacingRule	The rule used to determine line spacing: 0 (zero) or omitted Single 1 1.5 Lines 2 Double 3 At Least 4 Exactly 5 Multiple If you specify Single, 1.5 Lines, or Double, and also specify a value for .LineSpacing, the .LineSpacing value takes precedence.
.LineSpacing	The spacing for all lines within the paragraph (used when .LineSpacingRule is At Least, Exactly, or Multiple).
.Alignment	Sets a paragraph alignment: 0 (zero) Left 1 Centered 2 Right 3 Justified
.WidowControl	If 1, prevents a page break from leaving a single line of the paragraph at the top or bottom of a page (corresponds to selecting the Widow/Orphan Control check box).
.KeepWithNext	If 1, keeps the paragraph on the same page as the paragraph that follows (corresponds to selecting

	the Keep With Next check box).
.KeepTogether	If 1, keeps all lines in the paragraph on the same page (corresponds to selecting the Keep Lines Together check box).
.PageBreak	If 1, makes the paragraph always appear at the top of a new page (corresponds to selecting the Page Break Before check box).
.NoLineNum	If 1, turns off line numbering for the paragraph (corresponds to selecting the Suppress Line Numbers check box).
.DontHyphen	If 1, excludes the paragraph from automatic hyphenation (corresponds to selecting the Don't Hyphenate check box).
.Tab	Specifies which tab to select when you display the Paragraph dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) Indents And Spacing tab 1 Text Flow tab
.FirstIndent	The first-line indent in points or a text measurement.

See also

Paragraph Formatting Statements and Functions

EditFindPara

EditReplacePara

FormatBordersAndShading

FormatDefineStylePara

FormatStyle

FormatTabs

ParaKeepLinesTogether

ParaKeepWithNext

ParaPageBreakBefore

ParaWidowOrphanControl

Style

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatTabs Examples

This example sets a right-aligned tab stop at 1.5 inches:

```
FormatTabs .Position = "1.5 in", .Align = 2, .Set
```

The following example clears all custom tab stops:

```
FormatTabs .ClearAll
```

FormatTabs

¬Example

FormatTabs [**.Position =** *text*] [**, .DefTabs =** *number or text*] [**, .Align =** *number*] [**, .Leader =** *number*] [**, .Set**] [**, .Clear**] [**, .ClearAll**]

Sets and clears tab stops for the selected paragraphs. The arguments for the **FormatTabs** statement correspond to the options in the Tabs dialog box (Format menu).

Argument	Explanation
.Position	Position of the tab stop in a text measurement
.DefTabs	Position for default tab stops in the document in points or a text measurement
.Align	Alignment of the tab stop: 0 (zero) Left 1 Center 2 Right 3 Decimal 4 Bar
.Leader	The leader character for the tab stop: 0 (zero) None 1 Period 2 Hyphen 3 Underscore
.Set	Sets the specified custom tab stop
.Clear	Clears the specified custom tab stop
.ClearAll	Clears all custom tab stops

See also

Paragraph Formatting Statements and Functions

FormatDefineStyleTabs

NextTab()

PrevTab()

TabLeader$()

TabType()

HangingIndent

HangingIndent

Applies a hanging indent to the selected paragraphs, or increases the current hanging indent to the next tab stop of the first paragraph in the selection.

_____

See also

Paragraph Formatting Statements and Functions

Indent

UnHang

UnIndent

Indent

Indent

Moves the left indent of the selected paragraphs to the next tab stop of the first paragraph in the selection. **Indent** maintains the setting of a first-line or hanging indent.

_____

See also

Paragraph Formatting Statements and Functions

FormatParagraph

HangingIndent

UnIndent

InsertPara

InsertPara

Inserts a paragraph mark at the insertion point.

_____

See also

Paragraph Formatting Statements and Functions

Chr$()

Insert

JustifyPara, JustifyPara()

JustifyPara

JustifyPara()

The **JustifyPara** statement justifies the selected paragraphs. The **JustifyPara()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is justified
-1	If part of the selection is justified or there is a mix of alignments
1	If all the selection is justified

See also

Paragraph Formatting Statements and Functions

CenterPara

FormatParagraph

LeftPara

RightPara

LeftPara, LeftPara()

LeftPara

LeftPara()

The **LeftPara** statement aligns the selected paragraphs with the left indent.

The **LeftPara()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is left-aligned
-1	If part of the selection is left-aligned
1	If all the selection is left-aligned

See also

Paragraph Formatting Statements and Functions

CenterPara

FormatParagraph

JustifyPara

RightPara

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

NextTab() Example

This example uses **NextTab()** with **TabType()** to return a value corresponding to the alignment of the first tab stop in the selection:

```
firstTabType = TabType(NextTab(0))
```

NextTab()

¬Example

NextTab(*Position***)**

Returns the position of the next tab stop to the right of *Position*, in points, for the first paragraph of the selection.

Use the following list of conversions as an aid in converting from points to other measurements:

- ¬ 1 inch = 72 points
- ¬ 1 centimeter = 28.35 points
- ¬ 1 pica = 12 points

_____

See also

Paragraph Formatting Statements and Functions

FormatTabs

PrevTab()

TabLeader$()

TabType()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OpenUpPara Example

If there is no space before the current paragraph, this example sets the Before option to "12 pt":

```
Dim dlg As FormatParagraph
GetCurValues dlg
If dlg.Before = "0 pt" Then OpenUpPara
```

OpenUpPara

¬Example

OpenUpPara

Sets the Before option on the Indents And Spacing tab in the Paragraph dialog box (Format menu) to "12 pt."

_____

See also

Paragraph Formatting Statements and Functions

CloseUpPara

FormatParagraph

ParaKeepLinesTogether, ParaKeepLinesTogether()

ParaKeepLinesTogether [*On*]

ParaKeepLinesTogether()

The **ParaKeepLinesTogether** statement adds or removes the Keep Lines Together paragraph format for the selected paragraphs. All lines in a paragraph formatted with Keep Lines Together remain on the same page when Word repaginates the document.

Argument	Explanation
On	Specifies whether to add or remove the Keep Lines Together format: 1 Adds the Keep Lines Together format. 0 (zero) Removes the Keep Lines Together format. Omitted Toggles the Keep Lines Together format.

The **ParaKeepLinesTogether()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selected paragraphs are formatted with Keep Lines Together
-1	If some of the selected paragraphs are formatted with Keep Lines Together
1	If all the selected paragraphs are formatted with Keep Lines Together

See also

Paragraph Formatting Statements and Functions

FormatParagraph

ParaKeepWithNext

ParaPageBreakBefore

ParaKeepWithNext, ParaKeepWithNext()

ParaKeepWithNext [*On*]

ParaKeepWithNext()

The **ParaKeepWithNext** statement adds or removes the Keep With Next paragraph format for the selected paragraphs. A paragraph formatted with Keep With Next remains on the same page as the paragraph it follows when Word repaginates the document.

Argument	Explanation
On	Specifies whether to add or remove the Keep With Next format: 1 Adds the Keep With Next format. 0 (zero) Removes the Keep With Next format. Omitted Toggles the Keep With Next format.

The **ParaKeepWithNext()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selected paragraphs are formatted with Keep With Next
-1	If some of the selected paragraphs are formatted with Keep With Next
1	If all the selected paragraphs are formatted with Keep With Next

See also

Paragraph Formatting Statements and Functions

FormatParagraph

ParaKeepLinesTogether

ParaPageBreakBefore

ParaPageBreakBefore, ParaPageBreakBefore()

ParaPageBreakBefore [*On*]

ParaPageBreakBefore()

The **ParaPageBreakBefore** statement adds or removes the Page Break Before paragraph format for the selected paragraphs. A paragraph formatted with Page Break Before appears at the top of a new page whenever the document is printed.

Argument	Explanation
On	Specifies whether to add or remove the Page Break Before format: 1 Adds the Page Break Before format. 0 (zero) Removes the Page Break Before format. Omitted Toggles the Page Break Before format.

The **ParaPageBreakBefore()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selected paragraphs are formatted with Page Break Before
-1	If some of the selected paragraphs are formatted with Page Break Before
1	If all the selected paragraphs are formatted with Page Break Before

See also

Paragraph Formatting Statements and Functions

FormatParagraph

ParaKeepLinesTogether

ParaKeepWithNext

ParaWidowOrphanControl, ParaWidowOrphanControl()

ParaWidowOrphanControl [*On*]

ParaWidowOrphanControl()

The **ParaWidowOrphanControl** statement adds or removes the Widow/Orphan Control paragraph format for the selected paragraphs. A paragraph formatted with Widow/Orphan Control prevents a page break from leaving a single line of the paragraph at the top or bottom of a page.

Argument	Explanation
On	Specifies whether to add or remove the Widow/Orphan Control format: 1 Adds the Widow/Orphan Control format. 0 (zero) Removes the Widow/Orphan Control format. Omitted Toggles the Widow/Orphan Control format.

The **ParaWidowOrphanControl()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selected paragraphs are formatted with Widow/Orphan Control
-1	If some of the selected paragraphs are formatted with Widow/Orphan Control
1	If all the selected paragraphs are formatted with Widow/Orphan Control

See also

Paragraph Formatting Statements and Functions

FormatParagraph

ParaKeepLinesTogether

ParaKeepWithNext

ParaPageBreakBefore

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

PrevTab() Example

This example displays a message box stating the position, in inches, of the last tab stop in the current paragraph. In the first instruction, PrevTab(8.5 * 72) returns the position of the last tab in points, given a page width of 8.5 inches and 72 points per inch. The value is divided by 72 to give the position in inches. The next three instructions prepare the number for display in a message box, with a maximum of two digits after the decimal point.

```
LastPos = PrevTab(8.5 * 72) / 72
LastPos$ = Str$(LastPos)
dot = InStr(LastPos$, ".")
LastPos$ = Left$(LastPos$, dot + 2)
MsgBox "Position of last tab stop: " + LastPos$ + " inches"
```

PrevTab()

¬Example

PrevTab(*Position***)**

Returns the position of the next tab stop to the left of *Position*, in points, for the first paragraph in the selection.

Use the following list of conversions as an aid in converting from points to other measurements:

¬ 1 inch = 72 points
¬ 1 centimeter = 28.35 points
¬ 1 pica = 12 points

See also

Paragraph Formatting Statements and Functions

FormatTabs

NextTab()

TabLeader$()

TabType()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ResetPara Example

If the current paragraph is formatted as Normal, this example removes any paragraph formatting not defined as part of the style:

```
If StyleName$() = "Normal" Then ResetPara
```

ResetPara, ResetPara()

¬Example

ResetPara

ResetPara()

The **ResetPara** statement removes paragraph formatting not defined in the current paragraph style from the selected text. For example, if you format a paragraph with a half-inch left indent and the style applied to the paragraph has no left indent, **ResetPara** would remove the indent.

The **ResetPara()** function returns the following values without removing paragraph formatting.

Value	Explanation
0 (zero)	If the first paragraph in the selection contains formatting that differs from the formatting defined for the applied paragraph style
-1	If the first paragraph in the selection contains no formatting that differs from the formatting defined for the applied paragraph style

See also

Paragraph Formatting Statements and Functions

FormatParagraph

NormalStyle

ResetChar

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

RightPara Example

This example formats each right-aligned paragraph with a left indent of 2 inches:

```
StartOfDocument
While ParaDown()                        'Check each paragraph one by one,
    ParaUp                              'including the first paragraph
    If RightPara() = 1 Then             'If right-aligned, then indent left
        FormatParagraph .LeftIndent = "2 in"
    End If
    ParaDown
Wend
```

RightPara, RightPara()

¬Example

RightPara

RightPara()

The **RightPara** statement aligns the selected paragraphs with the right indent.

The **RightPara()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is right-aligned
-1	If part of the selection is right-aligned
1	If all the selection is right-aligned

See also

Paragraph Formatting Statements and Functions

CenterPara

FormatParagraph

JustifyPara

LeftPara

SpacePara1, SpacePara1()

SpacePara1

SpacePara1()

The **SpacePara1** statement single-spaces the selected paragraphs. The exact spacing is determined by the font size of the largest characters in the paragraph.

The **SpacePara1()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is single-spaced
-1	If part of the selection is single-spaced
1	If all the selection is single-spaced

See also

Paragraph Formatting Statements and Functions

CloseUpPara

FormatParagraph

OpenUpPara

SpacePara15

SpacePara2

SpacePara15, SpacePara15()

SpacePara15

SpacePara15()

The **SpacePara15** statement formats the selected paragraphs with 1.5 line spacing. The exact spacing is determined by the font size of the largest characters in the paragraph plus 6 points.

The **SpacePara15()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is formatted with 1.5 line spacing
-1	If part of the selection is formatted with 1.5 line spacing
1	If all the selection is formatted with 1.5 line spacing

See also

Paragraph Formatting Statements and Functions

CloseUpPara

FormatParagraph

OpenUpPara

SpacePara1

SpacePara2

SpacePara2, SpacePara2()

SpacePara2

SpacePara2()

The **SpacePara2** statement double-spaces the selected paragraphs. The exact spacing is determined by the point size of the largest characters in the paragraph plus 12 points.

The **SpacePara2()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selection is double-spaced
-1	If part of the selection is double-spaced
1	If all the selection is double-spaced

See also

Paragraph Formatting Statements and Functions

CloseUpPara

FormatParagraph

OpenUpPara

SpacePara1

SpacePara15

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TabLeader$() Example

This example determines whether a tab stop at the three-fourths-inch position has an underscore leader character. If it does, the **FormatTabs** instruction changes the leader character to "None."

```
ViewNormal
If TabLeader$(54) = "_" Then
    FormatTabs .Position = "54 pt", .Leader = 0
End If
```

TabLeader$()

¬Example

TabLeader$(*Pos***)**

Returns the leader character of the custom tab stop at the position *Pos*, which is given in points (72 points = 1 inch). If more than one paragraph is selected, **TabLeader$()** evaluates the settings in the first paragraph.

There are three cases in which **TabLeader$()** returns an empty string (""): if there is no tab stop at the position *Pos*, if the tab stop is a default tab stop, or if the document is in outline view. If there is a custom tab stop at *Pos*, **TabLeader$()** returns one of the following values.

Value	Explanation
(space)	No leader (corresponds to a tab leader set to "None")
.	Period
-	Hyphen
_	Underscore

To change the tab leader, use **FormatTabs**.

See also

Paragraph Formatting Statements and Functions

FormatTabs

NextTab()

PrevTab()

TabType()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TabType() Example

This example determines whether a tab stop at the three-fourths - inch position is centered. If it is, the **FormatTabs** instruction changes the alignment to left alignment.

```
If TabType(54) = 1 Then
    FormatTabs .Position = "54 pt", .Align = 0
End If
```

TabType()

¬Example

TabType(*Pos***)**

Returns the alignment of the custom or default tab stop at the position *Pos*, which is given in points (72 points = 1 inch). If more than one paragraph is selected, **TabType()** evaluates the setting in the first paragraph.

If there is no tab stop at the position *Pos*, **TabType()** returns -1. If there is a tab stop at *Pos*, **TabType()** returns one of the following values.

Value	Explanation
0 (zero)	Left-aligned or default
1	Centered
2	Right-aligned
3	Decimal
4	Bar

See also

Paragraph Formatting Statements and Functions

FormatTabs

NextTab()

PrevTab()

TabLeader$()

UnHang

UnHang

Removes a hanging indent from the selected paragraphs, or reduces the current hanging indent to the previous tab stop of the first paragraph in the selection.

See also

Paragraph Formatting Statements and Functions

HangingIndent

UnIndent

UnIndent

UnIndent

Moves the left indent of the selected paragraphs to the previous tab stop of the first paragraph in the selection. **UnIndent** maintains the setting of a first-line or hanging indent.

_____

See also

Paragraph Formatting Statements and Functions

Indent

UnHang

CountToolsGrammarStatistics()

CountToolsGrammarStatistics()

Returns the number of statistics that are stored when you check grammar with the **ToolsGrammar** statement. You can use this number to define the size of a two-dimensional array that you fill using the **ToolsGrammarStatisticsArray** statement.

For an example, see ToolsGrammarStatisticsArray Example.

See also

Proofing Statements and Functions

ToolsGrammar

ToolsGrammarStatisticsArray

ToolsOptionsGrammar

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsGetSpelling Example

This example checks the spelling of the word "kolor," and then displays up to five suggested replacements in a series of message boxes:

```
Dim suggest$(4)
ToolsGetSpelling suggest$(), "kolor"
For count = 0 To 4
    If suggest$(count) <> "" Then MsgBox suggest$(count)
Next
```

ToolsGetSpelling, ToolsGetSpelling()

¬Example

ToolsGetSpelling *FillArray$***()** [**,** *Word$*] [**,** *MainDic$*] [**,** *SuppDic$*] [**,** *Mode*]

ToolsGetSpelling(*FillArray$***()** [**,** *Word$*] [**,** *MainDic$*] [**,** *SuppDic$*] [**,** *Mode*]**)**

The **ToolsGetSpelling** statement fills an array with the words suggested as replacements for a misspelled word. Suggestions are assigned to elements of the array in the order they appear in the spelling checker.

Argument	Explanation
FillArray$	The array --- which must be defined before **ToolsGetSpelling** is run --- to fill with suggested replacements.
Word$	The word for which you want suggested replacements. If *Word$* is omitted, Word uses the word closest to the insertion point.
MainDic$	A text string that represents the language name for the main dictionary in the specified language. For a list of valid foreign language names, see ToolsLanguage. If you specify neither a word nor a main dictionary, Word uses the main dictionary that corresponds to the language formatting of the word closest to the insertion point.
SuppDic$	A path and filename for a custom dictionary (for example, "C:\WINWORD\USER1.DIC"). If *Word$* is found in the custom dictionary, **ToolsGetSpelling** puts no words in the array. If .SuggestFromMainDictOnly (**ToolsOptionsSpelling** statement) is 0 (zero), suggested replacements are retrieved from the custom dictionary as well as the main dictionary. Note that anagrams are retrieved from the main dictionary only.
Mode	Specifies how Word makes suggestions for *Word$*: 0 or omitted — Returns suggested correct spellings for *Word$*. 1 — Returns suggested replacements that match the search criteria when *Word$* contains the question mark (?) and asterisk (*) wildcard characters. 2 — Returns anagrams for *Word$* (words composed of the same letters as *Word$*). Word does not return anagrams from a custom dictionary. Note that this functionality may not

	be available if you are using a spelling checker other than the one shipped with Word.

The **ToolsGetSpelling()** function returns the number of replacements suggested by the spelling checker. If the word is spelled correctly, 0 (zero) is returned.

See also

Proofing Statements and Functions

ToolsGetSynonyms

ToolsLanguage

ToolsOptionsSpelling

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsGetSynonyms Example

This example prompts the user for a word, and then displays up to five synonyms for that word in a message box:

```
word$ = InputBox$("Display synonyms for: ", "Get Synonyms")
Dim synonyms$(4)
ToolsGetSynonyms synonyms$(), word$
list$ = synonyms$(0)
For count = 1 To 4
    If synonyms$(count) <> "" Then
        list$ = list$ + ", " + synonyms$(count)
    End If
Next
MsgBox list$, "Synonyms for " + word$
```

ToolsGetSynonyms, ToolsGetSynonyms()

¬Example

ToolsGetSynonyms *FillArray$***()** [**,** *Word$*] [**,** *MainDic$*]

ToolsGetSynonyms(*FillArray$***()** [**,** *Word$*] [**,** *MainDic$*]**)**

The **ToolsGetSynonyms** statement fills an array with synonyms for a word.

Argument	Explanation
FillArray$	The array --- which must be defined before **ToolsGetSynonyms** is run --- to fill with synonyms.
Word$	The word for which you want synonyms. If *Word$* is omitted, Word uses the word closest to the insertion point.
MainDic$	A text string that represents the language name for the main dictionary in the specified language. For a list of valid foreign language names, see ToolsLanguage. If you specify neither a word nor a main dictionary, Word uses the main dictionary that corresponds to the language formatting of the word closest to the insertion point.

The **ToolsGetSynonyms()** function returns the following values.

Value	Explanation
0 (zero)	If there are no synonyms available
-1	If one or more synonyms are available

See also

Proofing Statements and Functions

ToolsGetSpelling

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsGrammar Example

This example displays a message box that asks whether or not to check grammar. If the user chooses the Yes button (returning a value of -1), Word begins checking grammar.

```
ans = MsgBox("Check grammar now?", 3)
If ans = -1 Then ToolsGrammar
```

ToolsGrammar

¬Example

ToolsGrammar

Displays the Grammar dialog box (Tools menu) and begins checking grammar in the active document.

_____

See also

Proofing Statements and Functions

ToolsSpelling

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsGrammarStatisticsArray Example

This example defines a two-dimensional array for storing grammar statistics, fills the array with grammar statistics for the active document, and then inserts a two-column list of statistics at the insertion point (names of counts, averages, and indexes on the left and their associated statistics on the right, separated by tab characters):

```
numstats = CountToolsGrammarStatistics()
size = numstats -1
Dim docstats$(size, 1)
ToolsGrammarStatisticsArray docstats$()
For count = 0 To size
    InsertPara
    Insert docstats$(count, 0) + Chr$(9) + docstats$(count, 1)
Next
```

ToolsGrammarStatisticsArray

¬Example

ToolsGrammarStatisticsArray *TwoDimensionalArray$()*

Performs a grammar check, and then fills a two-dimensional array with the available grammar statistics for the active document.

Argument	Explanation
TwoDimensionalArray$()	The two-dimensional array --- which must be defined before **ToolsGrammarStatisticsArray** is run --- to fill with the names of counts, averages, and indexes (the first dimension) and their associated statistics (the second dimension).

See also

Proofing Statements and Functions

CountToolsGrammarStatistics()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsGrammar Example

This example specifies that spelling be checked with grammar, and then starts the grammar checker:

```
ToolsOptionsGrammar .CheckSpelling = 1, .ShowStatistics = 1
ToolsGrammar
```

ToolsOptionsGrammar

¬Example

ToolsOptionsGrammar [**.Options =** *number*] [**.CheckSpelling =** *number*] [**, .ShowStatistics =** *number*]

Sets grammar checking options. The arguments for the **ToolsOptionsGrammar** statement correspond to the options on the Grammar tab in the Options dialog box (Tools menu).

Argument	Explanation
.Options	Specifies which set of grammar rules to use: 0 (zero) All rules 1 Rules for business writing 2 Rules for casual writing 3 First custom settings 4 Second custom settings 5 Third custom settings
.CheckSpelling	If 1, checks spelling during grammar checking.
.ShowStatistics	If 1, readability statistics are displayed when the grammar check is complete.

For more information on the options described above, select the Grammar tab in the Options dialog box, and then choose the Help button.

See also

Proofing Statements and Functions

ToolsGrammar

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsSpelling Example

This example performs a spelling check in which words in all capital letters and words that contain numbers are not checked. The custom dictionary MYTERMS.DIC is used with the main dictionary.

```
ToolsOptionsSpelling .IgnoreAllCaps = 1, .IgnoreMixedDigits = 1,\
        .Type = 0, .CustomDict1 = "C:\WINWORD\MYTERMS.DIC"
ToolsSpelling
```

ToolsOptionsSpelling

¬Example

ToolsOptionsSpelling [**.AlwaysSuggest** = *number*] [**, .SuggestFromMainDictOnly** = *number*] [**, .IgnoreAllCaps** = *number*] [**, .IgnoreMixedDigits** = *number*] [**, .ResetIgnoreAll**] [**, .Type** = *number*] [**, .CustomDict***n* = *text*]

Sets options for checking spelling in a document. The arguments for the **ToolsOptionsSpelling** statement correspond to the options on the Spelling tab in the Options dialog box (Tools menu).

Argument	Explanation
.AlwaysSuggest	If 1, always suggests a replacement spelling for each misspelled word.
.SuggestFromMainDictOnly	If 1, draws spelling suggestions from the main dictionary only.
.IgnoreAllCaps	If 1, ignores words in all capital letters.
.IgnoreMixedDigits	If 1, ignores words that contain numbers.
.ResetIgnoreAll	Resets the Ignore All list so that Word will not ignore words for which you chose Ignore All while checking spelling during the current Word session.
.Type	Type of dictionary being searched: 0 (zero) Normal 2 Complete 3 Medical 4 Legal Note that if you specify 2, 3, or 4 and have not installed the corresponding dictionary, an error occurs. For information on how to purchase supplemental dictionaries, contact Microsoft Customer Service.
.CustomDict*n*	The path and filename of a custom dictionary to create or add; include a separate .CustomDict*n* argument for each custom dictionary, up to 10.

For more information on the options described above, select the Spelling tab in the Options dialog box, and then choose the Help button.

See also

Proofing Statements and Functions

ToolsSpelling

ToolsSpellSelection

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsSpelling Example

This example asks whether or not to check spelling, and then checks spelling if the user chooses the Yes button:

```
ans = MsgBox("Check spelling now?", 3)
If ans = -1 Then
    StartOfDocument
    ToolsSpelling
End If
```

ToolsSpelling

¬Example

ToolsSpelling

Checks spelling in the current selection or, if there isn't a selection, checks spelling from the location of the insertion point to the end of the document.

_____

See also

Proofing Statements and Functions

ToolsOptionsSpelling

ToolsSpellSelection

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsSpellSelection Example

This example asks whether or not to check the spelling of the current selection, and then checks spelling if the user chooses the Yes button:

```
ans = MsgBox("Check spelling of selection?", 3)
If ans = -1 Then ToolsSpellSelection
```

ToolsSpellSelection

¬Example

ToolsSpellSelection

Checks spelling in the current selection. If the selection is only part of a word, or if the insertion point is at the end of a word, the selection is extended to include the whole word. If the insertion point isn't in or doesn't follow a word, the next word is checked.

See also

Proofing Statements and Functions

ToolsOptionsSpelling

ToolsSpelling

ToolsThesaurus

ToolsThesaurus

Displays the Thesaurus dialog box (Tools menu), which lists alternatives for the selected word.

_____

See also

Proofing Statements and Functions

ToolsGetSynonyms

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatAutoFormat Example

This example automatically formats the active document using styles in REPORT1.DOT.

```
FormatStyleGallery .Template = "REPORT1"
FormatAutoFormat
```

FormatAutoFormat

¬Example

FormatAutoFormat

Automatically formats a document according to the options set with **ToolsOptionsAutoFormat**. To control the look of the document, you can use **FormatStyleGallery** to copy styles into the active document before or after running **FormatAutoFormat**.

See also

Section and Document Formatting Statements and Functions

FormatStyleGallery

ToolsOptionsAutoFormat

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatColumns Example

This example formats the current section into three columns of equal width and specifies that a vertical line appear between columns:

```
FormatColumns .Columns = "3", .EvenlySpaced = 1, .ColLine = 1
```

FormatColumns

¬Example

FormatColumns [**.Columns** = *number or text*] [**, .ColumnNo** = *text*] [**, .ColumnWidth** = *text*] [**, .ColumnSpacing** = *text*] [**, .EvenlySpaced** = *number*] [**, .ApplyColsTo** = *number*] [**, .ColLine** = *number*] [**, .StartNewCol** = *number*]

Formats all or part of the document with the specified number of columns, or changes the width of and space between existing columns. The arguments for the **FormatColumns** statement correspond to the options in the Columns dialog box (Format menu).

Argument	Explanation
.Columns	The number of columns to apply to the portion of the document specified by .ApplyColsTo.
.ColumnNo	If .EvenlySpaced is set to 0 (zero), the number of the column to format; if you are returning information from a **FormatColumns** dialog record, the column for which you want to return the width. To set the width of a column independently of other columns, you need to include an additional **FormatColumns** instruction to set the width of the last column. If you include only one **FormatColumns** instruction, the resulting number of columns will be the same as .ColumnNo. For example, if you change the width of the first column in a section containing three columns, there will be only one column after the instruction runs. To preserve the three-column format, you need to include an additional **FormatColumns** instructions and set .ColumnNo to 3.
.ColumnWidth	The width of the column specified by .ColumnNo, or if .EvenlySpaced is set to 1, the width of each column in the portion of the document specified by .ApplyColsTo.
.ColumnSpacing	The space between columns.
.EvenlySpaced	If 1, makes each column the same width.
.ApplyColsTo	Specifies the portion of the document to apply the column format to: 0 (zero) This section

	1 This Point Forward 2 Selected Sections 3 Selected Text 4 Whole Document
.ColLine	If 1, adds a line between columns.
.StartNewCol	If 1, corresponds to selecting the Start New Column check box (effective only if This Point Forward is specified by .ApplyColsTo).

See also

Section and Document Formatting Statements and Functions

TableColumnWidth

FormatHeaderFooterLink

FormatHeaderFooterLink

Replaces the current header or footer with the corresponding header or footer from the previous section. An error occurs if the previous section does not contain a corresponding header or footer.

See also

Section and Document Formatting Statements and Functions

ShowNextHeaderFooter

ShowPrevHeaderFooter

ToggleHeaderFooterLink

ViewHeader

FormatHeadingNumbering

FormatHeadingNumbering [**.Remove**] [**, .Preset =** *number*]

Adds or removes numbers for the headings in the selection. A heading is a paragraph formatted with one of the nine built-in heading styles.

Argument	Explanation
.Remove	Removes numbers from the headings in the selection.
.Preset	A number corresponding to a numbering scheme in the Heading Numbering dialog box (Format menu). To determine the appropriate number, open the Heading Numbering dialog box and determine the number of the scheme. From left to right, the preset schemes are numbered 1 through 6.

See also

Section and Document Formatting Statements and Functions

FormatBulletsAndNumbering

FormatPageNumber

FormatPageNumber [**.ChapterNumber** = *number*] [**, .NumRestart** = *number*] [**, .NumFormat** = *number*] [**, .StartingNum** = *number*] [**, .Level** = *number*] [**, .Separator** = *number*]

Determines the format of the page numbers used in the current section. The arguments for the **FormatPageNumber** statement correspond to the options in the Page Number Format dialog box (Page Numbers command, Insert menu).

Argument	Explanation
.ChapterNumber	If 1, includes the chapter number with the page number.
.NumRestart	Determines whether or not a different starting number can be set: 0 (zero) Selects the Continue From Previous Section option (the setting for .StartingNum has no effect). 1 Selects the Start At option (numbering begins at the number set for .StartingNum).
.NumFormat	The format for the page numbers: 0 (zero) 1 2 3... 1 a b c... 2 A B C... 3 i ii iii... 4 I II III...
.StartingNum	The current section's starting page number (if .NumRestart is set to 1).
.Level	A number corresponding to the heading level applied to the first paragraph in each chapter (for including the chapter number with the page number).
.Separator	If you specify .ChapterNumber, specifies the separator between the chapter number and the page number: 0 (zero) Hyphen 1 Period 2 Colon 3 Em dash 4 En dash

See also

Section and Document Formatting Statements and Functions

InsertPageNumbers

FormatSectionLayout

FormatSectionLayout [**.SectionStart =** *number*] [**, .VertAlign =** *number*] [**, .Endnotes =** *number*] [**, .LineNum =** *number*] [**, .StartingNum =** *number*] [**, .FromText =** *text*] [**, .CountBy =** *text*] [**, .NumMode =** *number*]

Applies section formatting to the selected sections. This command exists for compatibility with earlier versions of Word. The arguments for the **FormatSectionLayout** statement are duplicated in **FilePageSetup**.

See also

Section and Document Formatting Statements and Functions

FilePageSetup

ToggleHeaderFooterLink

ToggleHeaderFooterLink

If the current header or footer is not linked to the previous section, replaces the header or footer with the corresponding header or footer from the previous section and establishes a link. If the current header or footer is already linked, **ToggleHeaderFooterLink** breaks the link so you can modify the headers or footers independently. If the insertion point is not in a header or footer, an error occurs.

See also

Section and Document Formatting Statements and Functions

FormatHeaderFooterLink

ShowNextHeaderFooter

ShowPrevHeaderFooter

ViewHeader

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Abs() Example

This example toggles the display of hidden text. You can use this example as a model for toggling any check box in Word. In a Word dialog box, a selected check box has a value of 1; a cleared check box has a value of 0 (zero). The example reverses the current value of the check box by subtracting 1 from it and using **Abs()** to return the absolute value of the result.

```
Dim dlg As ToolsOptionsView
GetCurValues dlg
dlg.Hidden = Abs(dlg.Hidden - 1)
ToolsOptionsView dlg
```

Abs()

¬Example

Abs(*n*)

Returns the absolute value of *n*. For example, Abs(-5) returns the value 5.

_____

See also

Strings and Numbers Statements and Functions

Int()

Rnd()

Sgn()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Asc() Example

This example moves to the beginning of the next paragraph and checks the character to the right of the insertion point. If the character is a paragraph mark, indicating an empty paragraph, a message box is displayed.

```
ParaDown
If Asc(Selection$()) = 13 Then
    ans = MsgBox("Empty paragraph. Continue?", 4)
End If
```

Asc()

¬Example

Asc(*a$*)

Returns the character code of the first character in *a$*. Although "Asc" is short for ASCII, **Asc()** actually returns ANSI codes. The function name is preserved for compatibility purposes. You can use the syntax **Asc(Selection$())** to return the code for the character to the right of the insertion point (for example, to test for a paragraph mark).

See also

Strings and Numbers Statements and Functions

Chr$()

Len()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Chr$() Examples

This example displays a message box with two lines:

```
MsgBox "This is the first line" + Chr$(13) + "This is the second line"
```

The following example creates a table of symbols between ANSI 127 and ANSI 255 (often referred to as the extended character set). The first instruction asks the user to name the font for which he or she would like to see the symbols.

```
fontchoice$ = InputBox$("Please enter the Font Name ", \
                                        "Symbol Table", "Symbol")
For i = 127 To 255
    Font "Times New Roman"
    Insert Str$(i) + Chr$(9)
    Font fontchoice$
    Insert Chr$(i)
    InsertPara
Next
```

Chr$()

¬Example

Chr$(*CharCode***)**

Returns the character whose ANSI character code is *CharCode*.

Character codes in the range 0 (zero) to 31, inclusive, match the nonprinting characters of the standard ASCII code. For example, Chr$(13) is a carriage return character and Chr$(9) is a tab character. You can use Chr$(13) to create a new line within a message string used with **MsgBox** (but not **InputBox$()**).

The following table lists a few of the special characters you can produce using **Chr$()**.

Value	Character returned
Chr$(9)	Tab character
Chr$(11)	Newline character (SHIFT+ENTER)
Chr$(13)	Carriage return
Chr$(30)	Nonbreaking hyphen
Chr$(31)	Optional hyphen
Chr$(32)	Space character
Chr$(34)	Quotation mark
Chr$(160)	Nonbreaking space

The appearance of the symbol assigned to a given character code varies with the font used. Character codes in the range 127 to 255, inclusive, return different symbols, according to the font used.

Because the quotation mark is used to indicate the beginning or end of a string in WordBasic, you must use Chr$(34) if you want to include a quotation mark in a string. For example, to create a message box with the message "Type "Yes" or "No"," you would use the following statement:

```
MsgBox "Type " + Chr$(34) + "Yes" + Chr$(34) + " or " + \
    Chr$(34) + "No" + Chr$(34)
```

See also

Strings and Numbers Statements and Functions

Asc()

Str$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CleanString$() Example

This example uses **CleanString$()** to remove any nonprinting characters in the selected text:

```
temp$ = Selection$()
clean$ = CleanString$(temp$)
```

CleanString$()

¬Example

CleanString$(***Source$*)**

Changes nonprinting characters and special Word characters in *Source$* to spaces (ANSI character code 32). The following characters are changed to spaces, unless otherwise noted.

ANSI code	Description
1-29	Nonprinting characters. Character 13 (paragraph mark) is not removed. Character 10 is converted to character 13, unless preceded by character 13, in which case character 10 is removed. Character 7 is removed, unless preceded by character 13, in which case character 7 is converted to character 9 (tab character).
31	Nonprinting character. Character 31 is removed rather than changed to a space.
160	Nonbreaking space
172	Optional hyphen
176	Nonbreaking space
182	Paragraph mark character
183	Bullet character

If a field is included in the selection and the field codes are displayed, **CleanString$()** changes the field characters to spaces.

See also

Strings and Numbers Statements and Functions

LTrim$()

RTrim$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

InStr() Examples

This example sets the variable pos to 7:

```
list$ = "Bonn, Paris, Tokyo"
city$ = "Paris"
pos = InStr(list$, city$)
```

The following example sets pos to 15. Note that if *Index* were not specified, pos would be 2.

```
pos = InStr(3, "Bonn, Paris, Tokyo", "o")
```

The following example, which assigns a filename with a filename extension to name$, defines prefix$ as the filename without the extension:

```
name$ = "SUMMARY.DOC"
dot = InStr(name$, ".")
If dot > 1 Then prefix$ = Left$(name$, dot - 1)
```

InStr()

¬Example

InStr([*Index*,] *Source$*, *Search$***)**

Returns the character position in *Source$* at which *Search$* begins, where 1 corresponds to the first character, 2 to the second character, and so on. If *Source$* does not contain *Search$*, **InStr()** returns 0 (zero).

Argument	Explanation
Index	The character position in *Source$* at which to begin the search.
Source$	The text to be searched.
Search$	The text to search for.

See also

Strings and Numbers Statements and Functions

Left$()

Len()

LTrim$()

Mid$()

Right$()

RTrim$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Int() Examples

This example sets the variable x to 98:

```
x = Int(98.6)
```

The following example sets x to -9.

```
x = Int(-9.6)
```

Int()

¬Example

Int(*n*)

Returns the integer part of a decimal number *n*. If *n* is greater than or equal to 32,768 or less than or equal to -32,769, an error occurs.

_____

See also

Strings and Numbers Statements and Functions

Abs()

Rnd()

Sgn()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

LCase$() Example

This example displays the current directory in lowercase:

```
CurDir$ = Files$(".")
MsgBox LCase$(CurDir$)
```

LCase$()

¬Example

LCase$(***Source$*)**

Returns a string in which all letters of *Source$* have been converted to lowercase.

_____

See also

Strings and Numbers Statements and Functions

ChangeCase

UCase$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Left$() Examples

This example displays the text "Legal" in the status bar:

```
a$ = "Legal File List"
Print Left$(a$, 5)
```

The following example uses **Left$()** to return the first part of a hyphenated word. **InStr()** is used to determine the position of the hyphen (-) character. **Left$()** is then used to return all the characters to the left of the hyphen.

```
wholeWord$ = "fade-out"
hyphen = InStr(wholeWord$, "-")
firstWord$ = Left$(wholeWord$, (hyphen - 1))
MsgBox "First part of word: " + firstWord$
```

A similar set of instructions can be used to return the characters before the filename extension in an MS-DOS filename. Instead of determining the position of the hyphen, you would use **InStr()** to return the position of the period (.) character. This can be useful if you want to save a copy of the active document with a different filename extension. For an example, see InStr() Example.

Left$()

¬Example

Left$(***Source$*,** *Count***)**

Returns the leftmost *Count* characters of *Source$*.

See also

Strings and Numbers Statements and Functions

InStr()

Len()

LTrim$()

Mid$()

Right$()

RTrim$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Len() Example

This example prompts the user to type a name for an AutoText entry and then uses **Len()** to determine whether the name exceeds the 32-character limit. The **On Error** instruction prevents the "Command failed" message from appearing if the user cancels the **InputBox$()** dialog box.

```
On Error Goto Finish
Start:
a$ = InputBox$("Type a name for the AutoText entry.")
If Len(a$) > 32 Then
    MsgBox "Please type no more than 32 characters."
    Goto Start
End If
Finish:
```

Len()

¬Example

Len(*Source$***)**

Returns the number of characters in *Source$*.

_____

See also

Strings and Numbers Statements and Functions

InStr()

Left$()

LTrim$()

Mid$()

Right$()

RTrim$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

LTrim$() Example

This example converts the numeric variable code to a string and then removes the leading space that was automatically added by **Str$()**. The variable lastName$ is then joined with code$ and stored in the variable license$.

```
lastName$ = "Peterson"
code = 1234
code$ = Str$(code)
code$ = LTrim$(code$)
license$ = lastName$ + code$
```

LTrim$()

¬Example

LTrim$(***Source$*)**

Returns *Source$* minus any leading spaces (spaces to the left of the first character). **LTrim$()** is especially useful for removing the leading space from numeric values that have been converted to strings and for cleaning up user input.

See also

Strings and Numbers Statements and Functions

InStr()

Left$()

Mid$()

Right$()

RTrim$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Mid$() Examples

This example returns the second word of a two-word string:

```
wholeName$ = "Sanjeev Reddy"
space = InStr(wholeName$, " ")
lastName$ = Mid$(wholeName$, space + 1)
```

Mid$()

¬Example

Mid$(***Source$*,** *Start* [**,** *Count*]**)**

Returns a portion of *Source$* starting at a given character position.

Argument	Explanation
Source$	The original string.
Start	The character position in *Source$* where the string you want to return begins.
Count	The number of characters in the string you want to return. If you do not specify *Count*, the number of characters to the end of the string is assumed.

See also

Strings and Numbers Statements and Functions

InStr()

Left$()

Len()

LTrim$()

Right$()

RTrim$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Right$() Example

This example prompts the user to enter his or her first and last name in an **InputBox$()** dialog box. By subtracting the position of the space (located between the first and last name) from the length of the full name, the instructions determine the number of characters after the space. **Right$()** is then able to retrieve the last name from the single string.

```
fullName$ = InputBox$("Please type your first and last name.")
length = Len(fullName$)
space = InStr(fullName$, " ")
lastName$ = Right$(fullName$, length - space)
MsgBox "Last name is " + lastName$ + "."
```

Right$()

¬Example

Right$(***Source$*,** *Count***)**

Returns the rightmost *Count* characters of *Source$*.

_____

See also

Strings and Numbers Statements and Functions

InStr()

Left$()

Len()

LTrim$()

Mid$()

RTrim$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Rnd() Example

This example defines the variable num as a random integer between 50 and 100 and then displays the number in a message box:

```
a = 50
b = 100
num = Int(Rnd() * (b - a) + a)
MsgBox "The random number is" + Str$(num)
```

Rnd()

¬Example

Rnd()

Returns a random real number greater than or equal to 0 (zero) and less than 1. To generate a random integer between *a* and *b*, use the following syntax:

Int(Rnd() * (*b* - *a***) + *a***)**

See also

Strings and Numbers Statements and Functions

Abs()

Int()

Sgn()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

RTrim$() Example

This example prompts the user for his or her last name and then removes extra spaces the user may have typed at the end of the name:

```
lastName$ = InputBox$("Please enter your last name.")
lastName$ = RTrim$(lastName$)
```

RTrim$()

¬Example

RTrim$(***Source$*)**

Returns *Source$* minus any trailing spaces (spaces to the right of the last character). **RTrim$()** is especially useful for cleaning up user-defined variables before passing them to other parts of a macro.

_____

See also

Strings and Numbers Statements and Functions

InStr()

Left$()

LTrim$()

Mid$()

Right$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Selection$() Example

This example selects the first heading in the document and then uses the **Selection$()** function to define the variable heading$ with the selected text:

```
StartOfDocument
EditFindClearFormatting
EditFindStyle .Style = "heading 1"
EditFind .Find = "", .WholeWord = 0, .MatchCase = 0, \
    .Direction = 0, .Format = 1, .Wrap = 0
If EditFindFound() <> 0 Then
    heading$ = Selection$()
End If
```

Selection$()

¬Example

Selection$()

Returns the plain, unformatted text of the selection. **Selection$()** can return as many as 65,280 characters or the maximum that available memory can hold. If the selection is too large, an error occurs and no text is returned. If no text is selected, **Selection$()** returns the character following the insertion point.

See also

Strings and Numbers Statements and Functions

ExtendSelection

SelInfo()

SelType

ShrinkSelection

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Sgn() Example

This example compares two values. If the difference is negative, a message box is displayed.

```
maySales = 1200
juneSales = 1000
difference = juneSales - maySales
If Sgn(difference) = -1 Then
    MsgBox "Sales fell by" + Str$(Abs(difference)) + " dollars!"
End If
```

Sgn()

¬Example

Sgn(*n*)

Determines whether *n* is positive, negative, or 0 (zero).

Value	Explanation
0 (zero)	If *n* is 0 (zero)
-1	If *n* is a negative number
1	If *n* is a positive number

See also

Strings and Numbers Statements and Functions

Abs()

Int()

Rnd()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

SortArray Examples

This example creates an array containing the names of all the bookmarks in the active document and then sorts the names. When first defined, the variable marks$(0) represents the name of the first bookmark added to the document. After the array is sorted, marks$(0) represents the first name in an alphabetic list of the bookmark names.

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

The following example opens a text file containing a list of 100 names and addresses. There are five fields for the names and addresses: the first field is for the name, the second for the street address, the third for the city or town, the fourth for the state or province, and the fifth for the postal code. The array MailList$() is defined to accommodate the names and addresses, which are read into the array. The array is then sorted by postal code in descending order (so that rows with the highest postal code are first). The sorted names and addresses are then written back to the file. Note that you could perform the same sort by opening the file in a document window and using the **TableSort** statement.

```
Open "LIST.TXT" For Input As #1
Dim MailList$(99, 4)
For x = 0 To 99
    Read #1, MailList$(x, 0), MailList$(x, 1), MailList$(x, 2), \
        MailList$(x, 3), MailList$(x, 4)
Next
Close #1
SortArray MailList$(), 1, 0, 99, 0, 4
Open "NEWLIST.TXT" For Output As #1
For x = 0 To 99
    Write #1, MailList$(x, 0), MailList$(x, 1), MailList$(x, 2), \
                    MailList$(x, 3), MailList$(x, 4)
Next
Close #1
```

SortArray

¬Example

SortArray *ArrayName*[**$**]**()** [**,** *Order*] [**,** *From*] [**,** *To*] [**,** *SortType*] [**,** *SortKey*]

Performs an alphanumeric sort on the elements in the specified array. **SortArray** is especially useful for sorting arrays that fill list boxes in a custom dialog box. **SortArray** can sort one-dimensional or two-dimensional arrays; an error occurs if the specified array has more than two dimensions.

Argument	Explanation
ArrayName[**$**]**()**	The one-dimensional or two-dimensional array to be sorted. Arrays with more than two dimensions are not sorted.
Order	The sorting order: 0 (zero) or omitted Ascending 1 Descending
From	Number of the first element to sort. The default is 0 (zero).
To	Number of the last element to sort (must be greater than *From*).
SortType	The kind of sort to perform (applies only to two-dimensional arrays): 0 (zero) or omitted Sort the "rows" in the array matrix. 1 Sort the "columns" in the array matrix.
SortKey	The number of the row or column to sort by (applies only to two-dimensional arrays): 0 (zero) indicates the first row or column, 1 indicates the second, and so on. The default is 0 (zero). If *SortType* is 0 (zero), indicating a row sort, *SortKey* specifies the column that determines the sort. If *SortType* is 1, indicating a column sort, *SortKey* specifies the row that determines the sort.

Here are some examples of **SortArray** instructions, with descriptions of their effect.

Instruction	Description
SortArray ArrayTest()	Sort all the element(s) in the array ArrayTest() in ascending order, beginning with the element(s) numbered 0 (zero). If the array is two-dimensional, sort the rows of the array matrix, using the first column as the sort key.
SortArray List$(), 0, 1, 10	Sort the elements numbered 1 through 10 in the array List$() in ascending order.

SortArray MailingList$(), 1, 1, 20, 0, 1	Sort the elements numbered 1 through 20 in the two-dimensional array MailingList$() in descending order. Sort rows, using the second column as the sort key.
SortArray Table(), 0, 0, 10, 1, 3	Sort the elements numbered 0 (zero) through 10 in the two-dimensional array Table() in ascending order. Sort columns, using the fourth row as the sort key.

Note

Although the **SortArray** arguments other than *ArrayName*[**$**]**()** are optional, you cannot omit arguments between arguments that you do include. For example, SortArray Test(), 0, 0, 2, 0, 1 is a valid instruction, but SortArray Test(), 0, , , , 1 is not valid and will not sort the array.

See also

Strings and Numbers Statements and Functions

Dim

TableSort

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Str$() Examples

This example uses **Str$()** to make a numeric variable acceptable as part of a string in a message box. Note that no space is needed after "were" because **Str$()** has included a space at the beginning of a$.

```
sales = 2400
a$ = Str$(sales)
MsgBox "Sales this week were" + a$ + " dollars."
```

You can use the following function to return string representations of numeric variables without the leading space:

```
Function MyString$(Num)
    If Num >= 0 Then
        MyString$ = LTrim$(Str$(Num))
    Else
        MyString$ = Str$(Num)
    End If
End Function
```

In another subroutine, String1$ = MyString$(25) is an example of an instruction that calls the preceding function.

Str$()

¬Example

Str$(*n*)

Returns the string representation of the value *n*. If *n* is a positive number, **Str$()** returns a string with a leading space. To remove the leading space, use **LTrim$()**.

See also

Strings and Numbers Statements and Functions

Chr$()

InStr()

Left$()

LTrim$()

Mid$()

Right$()

RTrim$()

String$()

Val()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

String$() Examples

This instruction prints the text "ddddd" in the status bar:

```
Print String$(5, 100)
```

This example inserts a row of 40 characters with the ANSI character code 164 (¤) before the first selected paragraph:

```
SelType 1 : ParaDown : ParaUp                'Go to start of paragraph
Insert String$(40, 164)                      'Insert character string
InsertPara                                   'Insert a paragraph mark
```

String$()

¬Example

String$(*Count*, *Source$***)**

String$(*Count*, *CharCode***)**

Returns the first character in *Source$* or the character corresponding to the ANSI character code *CharCode*, repeated *Count* times. *Count* can be as large as 65,280.

_____

See also

Strings and Numbers Statements and Functions

Asc()

Chr$()

InStr()

Str$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

UCase$() Example

This example displays an **InputBox$** dialog box that prompts the user to type an acronym. When the user chooses OK, the text is stored in the variable acronym$, which is then converted to uppercase.

```
acronym$ = InputBox$("Please enter an acronym.")
acronym$ = UCase$(acronym$)
```

UCase$()

¬Example

UCase$(***Source$*)**

Returns a string in which all letters of *Source$* have been converted to uppercase.

See also

Strings and Numbers Statements and Functions

ChangeCase

LCase$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Val() Examples

In both of the following instructions, **Val()** returns 10:

```
num = Val("10")
num = Val("10 Apples")
```

In both of the following instructions, **Val()** returns 0 (zero):

```
num = Val("ten")
num = Val("Apartment 10")
```

The following example prompts the user to type a number, which the **InputBox$()** function returns as a string. **Val()** converts the string to a number and multiplies it by 12. **Str$()** converts the product to a string so that it can be displayed in a message box.

```
a$ = InputBox$("How many dozen apples?")
total = Val(a$) * 12
MsgBox a$ + " dozen equals" + Str$(total)
```

Val()

¬Example

Val(*a$*)

Returns the numeric value of *a$*. A common use of **Val()** is to convert strings containing digit characters to numbers so they may be used in mathematical formulas. If *a$* does not begin with a digit character, **Val()** returns 0 (zero).

See also

Strings and Numbers Statements and Functions

Str$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountStyles() Examples

This example returns the number of all built-in styles plus the number of user-created styles used in the template attached to the active document:

```
n = CountStyles(1, 1)
```

The following example merges styles into the active document from the attached template. Then, the macro finds the number of styles defined just for the document, if any, by subtracting the number of styles defined for the template from the number of styles defined for the document.

```
FormatStyle .Source = 1, .Merge
n = CountStyles(0, 0) - CountStyles(1, 0)
```

CountStyles()

¬Example

CountStyles([*Context*] [**,** *All*]**)**

Returns the number of styles defined for the specified context.

Argument	Explanation
Context	Specifies the location of styles to count: 0 (zero) or omitted Active document 1 Active template
All	Specifies whether to include built-in styles: 0 (zero) or omitted Built-in styles are excluded. 1 Built-in styles are included. Note that Word contains 75 built-in styles, and that two of those built-in styles are defined by default: Default Paragraph Font and Normal.

See also

Style Formatting Statements and Functions

StyleName$()

FormatDefineStyleBorders

FormatDefineStyleBorders [**.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 =** *number*]

Sets border and shading formats for either the current style or the style specified in a **FormatStyle** instruction containing the .Define argument. Note that when modifying the style specified with **FormatStyle**, the **FormatDefineStyleBorders** instruction follows the **FormatStyle** instruction.

The arguments for the **FormatDefineStyleBorders** statement correspond to the options in the Borders And Shading dialog box (Format menu). For argument descriptions, see FormatBordersAndShading. For examples of how to use similar statements to define a style, see FormatDefineStyleFont Example and FormatDefineStylePara Example.

See also

Style Formatting Statements and Functions

FormatBordersAndShading

FormatDefineStyleFont

FormatDefineStylePara

FormatStyle

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatDefineStyleFont Example

This example defines the character formatting of the "TestMe" style as 10-point, bold, and small caps. Word creates the style if it does not already exist.

```
FormatStyle .Name = "TestMe", .Define
FormatDefineStyleFont .Points = "10", .Bold = 1, .SmallCaps = 1
```

FormatDefineStyleFont

¬Example

FormatDefineStyleFont [**.Points** = *number*] [**, .Underline** = *number*] [**, .Color** = *number*] [**, .Strikethrough** = *number*] [**, .Superscript** = *value*] [**, .Subscript** = *value*] [**, .Hidden** = *number*] [**, .SmallCaps** = *number*] [**, .AllCaps** = *number*] [**, .Spacing** = *value*] [**, .Position** = *number or text*] [**, .Kerning** = *number*] [**, .KerningMin** = *number or text*] [**, .Default**] [**, .Tab** = *number*] [**, .Font** = *text*] [**, .Bold** = *number*] [**, .Italic** = *number*]

Sets character formats for either the current style or the style specified in a **FormatStyle** instruction containing the .Define argument. Note that when modifying the style specified with **FormatStyle**, the **FormatDefineStyleFont** instruction follows the **FormatStyle** instruction.

The arguments for the **FormatDefineStyleFont** statement correspond to options in the Font dialog box (Format menu). For argument descriptions, see FormatFont.

See also

Style Formatting Statements and Functions

FormatFont

FormatStyle

FormatDefineStyleFrame

FormatDefineStyleFrame [**.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**]

Sets frame formats for either the current style or the style specified in a **FormatStyle** instruction containing the .Define argument. Note that when modifying the style specified with **FormatStyle**, the **FormatDefineStyleFrame** instruction follows the **FormatStyle** instruction.

The arguments for the **FormatDefineStyleFrame** statement correspond to the options in the Frame dialog box (Format menu). For argument descriptions, see FormatFrame. For examples of how to use similar statements to define a style, see FormatDefineStyleFont Example and FormatDefineStylePara Example.

See also

Style Formatting Statements and Functions

FormatDefineStyleFont

FormatDefineStylePara

FormatFrame

FormatStyle

FormatDefineStyleLang

FormatDefineStyleLang .Language = *text* [**, .Default**]

Sets the language format for either the current style or the style specified in a **FormatStyle** instruction containing the .Define argument. Note that when modifying the style specified with **FormatStyle**, the **FormatDefineStyleLang** instruction follows the **FormatStyle** instruction.

The arguments for the **FormatDefineStyleLang** statement correspond to the options in the Language dialog box (Tools menu). For argument descriptions, see ToolsLanguage. For examples of how to use similar statements to define a style, see FormatDefineStyleFont Example and FormatDefineStylePara Example.

_____

See also

Style Formatting Statements and Functions

FormatDefineStyleFont

FormatDefineStylePara

FormatStyle

ToolsLanguage

FormatDefineStyleNumbers

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

Sets number formats for either the current style or the style specified in a **FormatStyle** instruction containing the .Define argument. Note that when modifying the style specified with **FormatStyle**, the **FormatDefineStyleNumbers** instruction follows the **FormatStyle** instruction.

The arguments for the **FormatDefineStyleNumbers** statement correspond to the options in the Modify List dialog box (Bullets And Numbering command, Format menu). For argument descriptions, see FormatBullet, FormatMultilevel, and FormatNumber. For examples of how to use similar statements to define a style, see FormatDefineStyleFont Example and FormatDefineStylePara Example.

_____

See also

Style Formatting Statements and Functions

FormatBullet

FormatDefineStyleFont

FormatDefineStylePara

FormatMultilevel

FormatNumber

FormatStyle

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatDefineStylePara Example

This example adds 4 points of Before formatting and 4 points of After formatting to the definition of the style "TestMe":

```
FormatStyle .Name = "TestMe", .Define
FormatDefineStylePara .Before = "4 pt", .After = "4 pt"
```

FormatDefineStylePara

¬Example

FormatDefineStylePara [**.LeftIndent =** *number or text*] [**, .RightIndent =** *number or text*] [**, .Before =** *number or text*] [**, .After =** *number or text*] [**, .LineSpacingRule =** *number*] [**, .LineSpacing =** *number or text*] [**, .Alignment =** *number*] [**, .WidowControl =** *number*] [**, .KeepWithNext =** *number*] [**, .KeepTogether =** *number*] [**, .PageBreak =** *number*] [**, .NoLineNum =** *number*] [**, .DontHyphen =** *number*] [**, .Tab =** *number*] [**, .FirstIndent =** *number or text*]

Sets paragraph formats for either the current style or the style specified in a **FormatStyle** instruction containing the .Define argument. Note that when modifying the style specified with **FormatStyle**, the **FormatDefineStylePara** instruction follows the **FormatStyle** instruction.

The arguments for the **FormatDefineStylePara** statement correspond to the options in the Paragraph dialog box (Format menu). For argument descriptions, see FormatParagraph.

See also

Style Formatting Statements and Functions

FormatParagraph

FormatStyle

FormatDefineStyleTabs

FormatDefineStyleTabs [**.Position =** *text*] [**, .DefTabs =** *number or text*] [**, .Align =** *number*] [**, .Leader =** *number*] [**, .Set**] [**, .Clear**] [**, .ClearAll**]

Sets tab formats for either the current style or the style specified in a **FormatStyle** instruction containing the .Define argument. Note that when modifying the style specified with **FormatStyle**, the **FormatDefineStyleTabs** instruction follows the **FormatStyle** instruction.

The arguments for the **FormatDefineStyleTabs** statement correspond to the options in the Tabs dialog box (Format menu). For argument descriptions, see FormatTabs. For examples of how to use similar statements to define a style, see FormatDefineStyleFont Example and FormatDefineStylePara Example.

See also

Style Formatting Statements and Functions

FormatDefineStyleFont

FormatDefineStylePara

FormatStyle

FormatTabs

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

FormatStyle Examples

This example applies the Normal style to the selected paragraphs and is equivalent to the **NormalStyle** statement:

```
FormatStyle .Name = "Normal", .Apply
```

The following example defines the character formatting of the "TestMe" style as 10-point, bold, and small caps. Word creates the style if it does not already exist.

```
FormatStyle .Name = "TestMe", .Define
FormatDefineStyleFont .Points = "10", .Bold = 1, .SmallCaps = 1
```

FormatStyle

¬Example

FormatStyle .Name = *text* [**, .Delete**] [**, .Merge**] [**, .NewName** = *text*] [**, .BasedOn =** *text*] [**, .NextStyle =** *text*] [**, .Type =** *number*] [**, .FileName =** *text*] [**, .Source** = *number*] [**, .AddToTemplate =** *number*] [**, .Define**] [**, .Rename**] [**, .Apply**]

Creates, modifies, redefines, applies, or deletes the specified style. You can also use **FormatStyle** to merge styles into the active document from its attached template. The arguments for the **FormatStyle** statement correspond to the options in the Style dialog box (Format menu).

To define formats for the style, follow the **FormatStyle** instruction with one or more of the following instructions: **FormatDefineStyleBorders**, **FormatDefineStyleFont**, **FormatDefineStyleFrame**, **FormatDefineStyleLang**, **FormatDefineStyleNumbers**, **FormatDefineStylePara**, **FormatDefineStyleTabs**.

To copy styles, use **Organizer**.

Argument	Explanation
.Name	The name of the style.
.Delete	Deletes the specified style.
.Merge	Merges styles to or from the document or template specified by .FileName, depending on the value specified for .Source.
.NewName	Specifies a new name for the style; used with .Rename.
.BasedOn	Specifies an existing style on which to base the specified style.
.NextStyle	Specifies the style to be applied automatically to the new paragraph after pressing ENTER in a paragraph formatted with the style specified by .Name.
.Type	When creating a new style, specifies the type: 0 (zero) or omitted Paragraph 1 Character
.FileName	The document or template to merge styles to or from. If .FileName is omitted and .Source is set to 1, styles are merged from the active template to the active document.
.Source	Specifies whether to merge styles to or from the active document: 0 (zero) From the active document to the file specified by .FileName. 1 From the file specified by .FileName to the active document.
.AddToTemplate	If 1, corresponds to selecting the Add To Template check box.
.Define	Redefines an existing style or creates a new style with the formats specified in subsequent

		instructions.
	.Rename	Renames the style specified by .Name to the name specified by .NewName.
	.Apply	Applies the style to the selected paragraphs.

See also

Style Formatting Statements and Functions

CountStyles()

FormatDefineStyleBorders

FormatDefineStyleFont

FormatDefineStyleFrame

FormatDefineStyleLang

FormatDefineStyleNumbers

FormatDefineStylePara

FormatDefineStyleTabs

NormalStyle

Organizer

Style

StyleName$()

FormatStyleGallery

FormatStyleGallery .Template = *text* [**, .Preview =** *number*]

Copies the styles from the specified template to the active document. Existing styles are updated.

Argument	Explanation
.Template	The template containing the styles you want to use. Paths are allowed. Including the .DOT filename extension is not required.
.Preview	Specifies what to preview in the StyleGallery dialog box when you display the dialog box with a **Dialog** or **Dialog()** instruction: 0 (zero) Active document 1 Example document 2 List of styles and samples

For an example, see FormatAutoFormat Example.

See also

Style Formatting Statements and Functions

FormatAutoFormat

FormatStyle

ToolsOptionsAutoFormat

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

NormalStyle Example

This example selects the current paragraph, and if it contains the string "Comments:", applies the Normal style. The bookmark "\Para" is one of several predefined bookmarks that Word sets and updates automatically. For information on predefined bookmarks, see Predefined Bookmarks.

```
EditGoTo .Destination = "\Para"
If InStr(Selection$(), "Comments:") Then NormalStyle
```

NormalStyle

¬Example

NormalStyle

Applies the Normal style to the selected paragraphs. There is not a corresponding function for **NormalStyle**; however, you can use the **StyleName$()** function to determine the current style. For example, the instruction `If StyleName$() = "Normal" Then MsgBox "Normal paragraph"` displays a message box if the current paragraph is in the Normal style.

_____

See also

Style Formatting Statements and Functions

FormatStyle

ResetPara

Style

StyleName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Style Example

This example finds the first instance of the word "Overview" followed by a paragraph mark and applies the Heading 1 style:

```
StartOfDocument
EditFind .Find = "Overview^p", .Direction = 0, .MatchCase = 1, \
    .Format = 0
If EditFindFound() Then Style "Heading 1"
```

Style

¬Example

Style *StyleName$*

Applies a style to the selected paragraphs. If *StyleName$* does not exist, an error occurs. To create a style, use **FormatStyle**.

_____

See also

Style Formatting Statements and Functions

FormatStyle

NormalStyle

StyleName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

StyleDesc$() Example

This example displays the description of the Heading 1 style in a message box:

```
infostyle$ = "Heading 1"
MsgBox StyleDesc$(infostyle$), "Description for " + infostyle$
```

StyleDesc$()

¬Example

StyleDesc$(***StyleName$*)**

Returns the description of the specified style in the active document. For example, a typical description for the Heading 1 style is "Normal + Arial, 14 pt, Bold, Space Before 12 pt After 3 pt." If *StyleName$* does not exist, **StyleDesc$()** returns an empty string ("").

See also

Style Formatting Statements and Functions

CountStyles()

FormatStyle

StyleName$()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

StyleName$() Example

This example checks to see if the first style in the list of document styles is used in the document. If the style is not used, a message box is displayed:

```
FirstStyle$ = StyleName$(1)
StartOfDocument
EditFindClearFormatting
EditFindStyle .Style = FirstStyle$
EditFind .Find = "", .WholeWord = 0, .MatchCase = 0, \
    .Direction = 0, .Format = 1, .Wrap = 0
If EditFindFound() = 0 Then
    MsgBox "Style " + FirstStyle$ + " is not in use."
End If
```

StyleName$()

¬Example

StyleName$([*Count***,**] [*Context***,**] [*All*]**)**

Returns the name of the specified style or, if no style is specified, the name of the style applied to the first paragraph in the selection.

Argument	Explanation
Count	The position of the style in an alphabetic list of styles for the document or template, according to the value of *Context*. The number can range from 1 to **CountStyles()**, which returns the total number of styles in the specified context. If *Count* is 0 (zero) or omitted, the name of the current style is returned.
Context	The list of styles to use: 0 (zero) or omitted Active document 1 Active template
All	Specifies whether to include built-in styles: 0 (zero) or omitted Built-in styles are excluded. 1 Built-in styles are included. Note that Word contains 75 built-in styles, and that two of those built-in styles are defined by default: Default Paragraph Font and Normal.

See also

Style Formatting Statements and Functions

CountStyles()

FormatStyle

StyleDesc$()

InsertExcelTable

InsertExcelTable

Starts Microsoft Excel and displays a new worksheet. When the user exits Microsoft Excel, the worksheet is embedded in the active document. The instruction InsertExcelTable is the same as the instruction InsertObject .Class = "ExcelWorksheet".

See also

Tables Statements and Functions

InsertChart

InsertDatabase

InsertObject

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableAutoFormat Example

This example applies all the properties of the Classic 2 table format to the table containing the insertion point. If the insertion point is not in a table, a message box is displayed.

```
If SelInfo(12) = -1 Then                    'If insertion point is in a table
    TableAutoFormat .Format = 5, .Borders = 1, .Shading = 1, \
    .Font = 1, .Color = 1, .AutoFit = 1, .HeadingRows = 1, \
    .FirstColumn = 1, .LastRow = 0, .LastColumn = 1
Else
    MsgBox "The insertion point is not in a table."
End If
```

TableAutoFormat

¬Example

TableAutoFormat [**.Format** = *number*] [**, .Borders** = *number*] [**, .Shading** = *number*] [**, .Font** = *number*] [**, .Color** = *number*] [**, .AutoFit** = *number*] [**, .HeadingRows** = *number*] [**, .FirstColumn** = *number*] [**, .LastRow** = *number*] [**, .LastColumn** = *number*]

Applies a predefined look to a table. The arguments for the **TableAutoFormat** statement correspond to the options in the Table AutoFormat dialog box (Table menu).

Argument	Explanation
.Format	The format to apply: 0 (zero) corresponds to the first format listed in the Formats list box ("none"), 1 corresponds to the second format, and so on.
.Borders	If 1, applies the border properties of the specified format.
.Shading	If 1, applies the shading properties of the specified format.
.Font	If 1, applies the font properties of the specified format.
.Color	If 1, applies the color properties of the specified format.
.AutoFit	If 1, decreases the width of the table columns as much as possible without changing the way text wraps in the cells.
.HeadingRows	If 1, applies the heading-row properties of the specified format.
.FirstColumn	If 1, applies the first-column properties of the specified format.
.LastRow	If 1, applies the last-row properties of the specified format.
.LastColumn	If 1, applies the last-column properties of the specified format.

See also

Tables Statements and Functions

TableColumnWidth

TableHeadings

TableRowHeight

TableUpdateAutoFormat

TableAutoSum

TableAutoSum

Inserts an = (Formula) field that calculates and displays the sum of the values in table cells above or to the left of the cell containing the insertion point. For information on how Word determines which values to add, see TableFormula.

_____

See also

Tables Statements and Functions

TableFormula

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableColumnWidth Example

For each table in the document, this example adjusts the width of the first column to 2 inches and the second column to 3 inches. The first **TableColumnWidth** instruction both formats the first column and moves the selection to the second column.

```
StartOfDocument
While ParaDown()
    If SelInfo(12) = -1 Then                    'If insertion point is in a table
        StartOfRow
        TableSelectColumn
        TableColumnWidth .ColumnWidth = "2 in", .RulerStyle = 0, \
                         .NextColumn
      If SelInfo(18) > 1 Then                   'If more than one column
                         TableColumnWidth .ColumnWidth = "3 in", .RulerStyle = 0
        End If
        LineDown
    End If
Wend
```

TableColumnWidth

¬Example

TableColumnWidth [**.ColumnWidth =** *number or text*] [**, .SpaceBetweenCols =** *number or text*] [**, .PrevColumn**] [**, .NextColumn**] [**, .AutoFit**] [**, .RulerStyle =** *number*]

Sets the column width and the space between columns for the selected cells. The arguments for the **TableColumnWidth** statement correspond to the options on the Column tab in the Cell Height And Width dialog box (Table menu).

Argument	Explanation
.ColumnWidth	The width to apply to the selected cells or columns, in points or a text measurement (.RulerStyle specifies how other columns are affected).
.SpaceBetweenCols	The distance between the text in each column, in points or a text measurement.
.PrevColumn	After the actions (if any) specified by the preceding arguments are carried out, selects the previous column.
.NextColumn	After the actions (if any) specified by the preceding arguments are carried out, selects the next column.
.AutoFit	Decreases the width of the selected cells as much as possible without changing the way text wraps in the cells.
.RulerStyle	Specifies how Word adjusts the table: 0 (zero) If there is a selection, only the selected cells are changed. Row width is not preserved. If there is no selection, Word sizes all cells in the selected column. 1 Word preserves row width by adjusting all cells to the right of the selection in proportion to their widths. 2 Word preserves row width by adjusting cells in the column immediately to the right of the selection only. 3 Word preserves row width by adjusting all cells to the right of the selection, assigning them each the same width. 4 Only the cell containing the insertion point (or the first cell in the selection) is changed. Row width is not preserved.

See also

Tables Statements and Functions

SelInfo()

TableDeleteColumn

TableRowHeight

TableSelectColumn

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableDeleteCells Example

This example deletes the first cell in a table if the table contains more than two rows:

```
If SelInfo(15) > 2 Then
    TableSelectTable
    StartOfRow
    TableDeleteCells
End If
```

TableDeleteCells

¬Example

TableDeleteCells .ShiftCells = *number*

Deletes the selected cells. If the insertion point or selection is not within a table, an error occurs.

Argument	Explanation
.ShiftCells	Sets the direction to shift the remaining cells: 0 (zero) or omitted Shift the cells left. 1 Shift the cells up. 2 Delete the entire row. 3 Delete the entire column.

See also

Tables Statements and Functions

SelInfo()

TableDeleteColumn

TableDeleteRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableDeleteColumn Example

This example deletes the first column in a table:

```
StartOfRow
TableDeleteColumn
```

TableDeleteColumn

¬Example

TableDeleteColumn

Deletes the table column containing the insertion point, or deletes all columns containing part of the selection. If the insertion point or selection is not within a table, an error occurs.

_____

See also

Tables Statements and Functions

SelInfo()

TableDeleteCells

TableDeleteRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableDeleteRow Example

This example deletes the last row in a table if the table contains more than two rows:

```
If SelInfo(15) > 2 Then
    EndOfColumn
    TableDeleteRow
End If
```

TableDeleteRow

¬Example

TableDeleteRow

Deletes the row containing the insertion point, or deletes all rows containing part of the selection. If the insertion point or selection is not within a table, an error occurs.

_____

See also

Tables Statements and Functions

SelInfo()

TableDeleteCells

TableDeleteColumn

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableFormula Example

This example adds a row to the end of the table, moves the insertion point to the last cell, and then inserts an = (Formula) field that adds the values in the last column:

```
TableSelectTable
EndOfRow
NextCell
EndOfRow
TableFormula .Formula = "=SUM(ABOVE)"
```

TableFormula

¬Example

TableFormula [**.Formula =** *text*] [**, .NumFormat =** *text*]

Inserts an = (Formula) field containing the specified formula at the insertion point. If the insertion point is in a table cell that already contains an = (Formula) field, the existing field is replaced with a field containing the specified formula.

Argument	Explanation
.Formula	The mathematical formula you want the = (Formula) field to evaluate. Spreadsheet-type references to table cells are valid. For example, "=SUM(a4:c4)" specifies the first three values in the fourth row. For more information on valid formulas, see = (Formula) field.
.NumFormat	A format for the result of the = (Formula) field. For sample formats, choose Formula from the Table menu and review the list in the Number Format box.

If the insertion point is in a table, specifying .Formula is optional so long as there is at least one cell containing a value above or to the left of the cell containing the insertion point. If the cells above contain values, the inserted field is **{**=SUM(ABOVE)**}**; if the cells to the left contain values, the inserted field is **{**=SUM(LEFT)**}**. If the cells above and the cells to the left contain values, Word uses the following rules to determine which SUM function to insert:

- If the cell immediately above the insertion point contains a value, Word inserts **{**=SUM(ABOVE)**}**.
- If the cell immediately above does not contain a value and the cell immediately to the left does, Word inserts **{**=SUM(LEFT)**}**.
- If neither adjoining cell contains a value, Word inserts **{**=SUM(ABOVE)**}**.

If you don't specify .Formula and all the cells above and to the left of the insertion point are empty, a field error occurs.

See also

Tables Statements and Functions

InsertField

TableAutoSum

ToolsCalculate

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableGridlines Example

This example displays table gridlines if they are not already showing. You could use this instruction as part of an AutoOpen macro, so that whenever documents based on a certain template are opened, table gridlines are displayed.

```
If Not TableGridlines() Then TableGridlines 1
```

TableGridlines, TableGridlines()

¬Example

TableGridlines [*On*]

TableGridlines()

The **TableGridlines** statement displays or hides table gridlines.

Argument	Explanation
On	Specifies whether to display or hide table gridlines: 0 (zero) Hides table gridlines. 1 Displays table gridlines. Omitted Toggles the option on or off.

The **TableGridlines()** function returns the following values.

Value	Explanation
0 (zero)	If table gridlines are hidden
-1	If table gridlines are displayed

See also

Tables Statements and Functions

ToolsOptionsView

TableHeadings, TableHeadings()

TableHeadings [*On*]

TableHeadings()

The **TableHeadings** statement adds or removes the table heading format for the selected rows. Rows formatted as table headings are repeated when a table spans more than one page.

Argument	Explanation
On	Specifies whether to add or remove the table heading format: 0 (zero) Removes the table heading format. 1 Adds the table heading format. Omitted Toggles the table heading format.

The **TableHeadings()** function returns the following values.

Value	Explanation
0 (zero)	If none of the selected rows are formatted as table headings
-1	If one or more but not all of the selected rows are formatted as table headings
1	If all the selected rows are formatted as table headings

See also

Tables Statements and Functions

TableRowHeight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableInsertCells Example

This example selects the current cell (using the predefined bookmark "\Cell") and the cell below it, and then inserts two new cells. The existing cells are shifted to the right. The **If** control structure makes sure the selection is entirely within a table before running **TableInsertCells**.

```
EditGoTo "\Cell"
LineDown 1, 1
If SelInfo(12) = -1 Then
    TableInsertCells .ShiftCells = 0
Else
    MsgBox "Selection extends outside the table."
End If
```

TableInsertCells

¬Example

TableInsertCells [**.ShiftCells =** *number*]

Inserts cells above or to the left of the selected range of cells in a table. If the insertion point or selection is not in a table, an error occurs.

Argument	Explanation
.ShiftCells	Sets the direction to shift the cells in the selected range: 0 (zero) Shift the cells right. 1 or omitted Shift the cells down. 2 Insert an entire row. 3 Insert an entire column.

See also

Tables Statements and Functions

SelInfo()

TableInsertColumn

TableInsertRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableInsertColumn Example

This example adds a column to the end of the table containing the insertion point:

```
TableSelectTable        'Select the entire table
StartOfColumn           'Move to the end of the first row
TableInsertColumn       'Add a new column
```

TableInsertColumn

¬Example

TableInsertColumn

Inserts a column to the left of the column that contains the insertion point, or inserts as many columns as are selected to the left of the selection. If the insertion point or selection is not in a table, an error occurs.

_____

See also

Tables Statements and Functions

TableInsertCells

TableInsertRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableInsertRow Examples

The following example adds a row to the end of a table:

```
TableSelectTable
CharRight
TableInsertRow
```

This example adds two new rows at the top of a table:

```
TableSelectTable
StartOfRow
TableInsertRow .NumRows = 2
```

TableInsertRow

¬Example

TableInsertRow [**.NumRows =** *number*]

Inserts a row above the selected rows or above the row that contains the insertion point. If the insertion point or any part of the selection is not in a table, an error occurs. If the insertion point immediately follows a table, **TableInsertRow** inserts a row at the end of the table.

Argument	Explanation
.NumRows	The number of rows you want to add. If .NumRows is 0 (zero) or omitted, **TableInsertRow** inserts above the selection as many rows as are selected.

See also

Tables Statements and Functions

TableInsertCells

TableInsertColumn

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableInsertTable Example

This example inserts a three-column, five-row table:

```
TableInsertTable .NumColumns = 3, .NumRows = 5, \
        .InitialColWidth = "2 in"
```

TableInsertTable

¬Example

TableInsertTable [**.ConvertFrom =** *number*] [**, .NumColumns =** *number*] [**, .NumRows =** *number*] [**, .InitialColWidth =** *number or text*] [**, .Wizard**] [**, .Format =** *number*] [**, .Apply =** *number*]

Converts a series of selected paragraphs into a table or inserts an empty table if there is no selection. If the insertion point is already in a table, an error occurs.

Argument	Explanation
.ConvertFrom	Specifies the character used to separate items of text into cell contents: 0 (zero) Paragraph marks (Word places every other paragraph in the second column) 1 Tab characters 2 Commas
.NumColumns	Number of columns in the table.
.NumRows	Number of rows in the table.
.InitialColWidth	The initial width for each column, in points or a text measurement. If omitted, column width is calculated so that the table stretches from margin to margin.
.Wizard	Runs the Table wizard.
.Format	One of the predefined formats listed in the Table AutoFormat dialog box (Table menu): 0 (zero) corresponds to the first format listed in the Formats box ("none"), 1 corresponds to the second format, and so on.
.Apply	Specifies which attributes of the format specified by .Format to apply to the table. Use the sum of any combination of the following values: 0 None 1 Borders 2 Shading 4 Font 8 Color 16 AutoFit 32 Heading Rows 64 Last Row 128 First Column 256 Last Column

See also

Tables Statements and Functions

TableAutoFormat

TableToText
TextToTable

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableMergeCells Example

This example either merges the currently selected cells or displays a message box if the command cannot be performed:

```
On Error Goto No
TableMergeCells
No:
If Err = 509 Then
    MsgBox "Sorry, can't merge."
End If
```

TableMergeCells

¬Example

TableMergeCells

Merges selected table cells in the same row into a single cell. Selections can extend over multiple rows; however, Word only merges cells in the same row. **TableMergeCells** generates an error if Word cannot merge cells --- for example, if only one cell is selected.

See also

Tables Statements and Functions

TableSplitCells

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableRowHeight Example

This example sets a minimum row height of 2 lines for the selected rows and indents them 1 inch from the left margin:

```
TableRowHeight .RulerStyle = "0", .LeftIndent = "1 in", \
    .LineSpacingRule = 1, .LineSpacing = "2 li", .Alignment = 0
```

TableRowHeight

¬Example

TableRowHeight [**.RulerStyle =** *text*] [**, .LineSpacingRule =** *number*] [**, .LineSpacing =** *number or text*] [**, .LeftIndent =** *number or text*] [**, .Alignment =** *number*] [**, .AllowRowSplit =** *number*] [**, .PrevRow**] [**, .NextRow**]

Sets formats for the selected rows in a table. The arguments for the **TableRowHeight** statement correspond to the options on the Row tab in the Cell Height And Width dialog box (Table menu).

Argument	Explanation
.RulerStyle	When the left indent is changed, specifies how Word adjusts the table: 0 (zero) Word moves cells to the right. 1 Word preserves the position of the right edge of the table by narrowing all cells in the selected rows in proportion to their widths. 2 Word preserves the position of the right edge of the table by narrowing cells in the first column only. 3 Word preserves the position of the right edge of the table by narrowing all cells in the selected rows, assigning them each the same width. 4 Word indents only the row containing the insertion point (or the first row in the selection). When row height is changed, specifies which rows are affected: 0 (zero), 1, 2, or 3 If there is a selection, only the selected rows are affected. If there is no selection, all rows are affected. 4 Only the row containing the insertion point or the first row in the selection is affected.
.LineSpacingRule	The rule for determining the height of the rows: 0 (zero) Auto 1 At Least 2 Exactly
.LineSpacing	The height of the rows, in points or a text measurement.
.LeftIndent	The distance between the left edge of the text and the left margin, in points or a text measurement.
.Alignment	The alignment of the rows: 0 (zero) Left 1 Center 2 Right
.AllowRowSplit	If 1, allows text in the row to be divided at a page break.
.PrevRow	Selects the previous row for

	formatting.
.NextRow	Selects the next row for formatting.

See also

Tables Statements and Functions

TableColumnWidth

TableHeadings

TableSelectRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableSelectColumn Example

This example adds two new columns before the first column in a table:

```
TableSelectTable          'Select the entire table
StartOfRow                'Go to the first cell
TableSelectColumn         'Select the first column
CharRight 1, 1            'Extend selection to the second column
TableInsertColumn         'Insert two new columns
```

TableSelectColumn

¬Example

TableSelectColumn

Selects the table column containing the insertion point, or selects all columns containing the selection. If the insertion point or selection is not in a table, an error occurs.

_____

See also

Tables Statements and Functions

TableSelectRow

TableSelectTable

TableSelectRow

TableSelectRow

Selects the table row containing the insertion point, or selects all rows containing the selection. If the insertion point or selection is not in a table, an error occurs.

_____

See also

Tables Statements and Functions

TableSelectColumn

TableSelectTable

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableSelectTable Examples

This example moves the insertion point to the first cell in a table:

```
TableSelectTable
StartOfRow
```

The following example moves the insertion point to the end-of-row mark in the first row in a table. When the insertion point is at this position, you can use **TableInsertColumn** to add a column to the end of a table.

```
TableSelectTable
StartOfColumn
```

The following example moves the insertion point to the first character after a table:

```
TableSelectTable
CharRight
```

TableSelectTable

¬Example

TableSelectTable

Selects the entire table containing the insertion point. If the insertion point or selection is not in a table, an error occurs. As the examples in this entry demonstrate, you can use **TableSelectTable** in combination with other WordBasic statements to move the insertion point reliably to the first cell in a table or to the first character after a table.

See also

Tables Statements and Functions

TableSelectColumn

TableSelectRow

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableSort Example

This example sorts table rows in ascending alphanumeric order, first by the first column, and then by the second column. The first row is excluded from the sort operation.

```
Select Case SelInfo(15)
    Case -1
        MsgBox "Please place the insertion point in a table."
    Case 1
        MsgBox "Sorry, can't sort a single-row table."
    Case Else
        TableSelectTable
        TableSort .Order = 0, .FieldNum = "1", .Type = 0, \
                              .Order2 = 0, .FieldNum2 = "2", .Type2 = 0, \
                              .DontSortHdr = 1
End Select
```

The following illustration shows a table before and after sorting by the preceding example.

Last	First
Quinn	Steve
Smith	Peter
Alicea	Juanita
Smith	Eileen
Alicea	Carlos

A table before sorting...

Last	First
Alicea	Carlos
Alicea	Juanita
Quinn	Steve
Smith	Eileen
Smith	Peter

and after sorting.

TableSort

¬Example

TableSort [**.DontSortHdr** = *number*] [**, .FieldNum** = *number*] [**, .Type** = *number*] [**.Order** = *number*] [**, .FieldNum2** = *number*] [**, .Type2** = *number*] [**, .Order2** = *number*] [**, .FieldNum3** = *number*] [**, .Type3** = *number*] [**, .Order3** = *number*] [**, .Separator** = *number*] [**, .SortColumn** = *number*] [**, .CaseSensitive** = *number*]

Sorts the selected paragraphs or table rows. If you want to sort paragraphs within a table cell, select only the paragraphs and not the end-of-cell mark; if you select the end-of-cell mark and then sort, Word displays a message stating that it found no valid records to sort. The arguments for the **TableSort** statement correspond to the options in the Sort dialog box (Table menu).

Argument	Explanation
.DontSortHdr	If 1, excludes the first paragraph or table row from the sort operation.
.FieldNum, .FieldNum2, .FieldNum3	The number of the fields (text or table columns) to sort by. Word sorts by .FieldNum, then by .FieldNum2, and then by .FieldNum3.
.Type, .Type2, .Type3	The respective sort types for .FieldNum, .FieldNum2, and .FieldNum3: 0 (zero) Alphanumeric 1 Numeric 2 Date
.Order, .Order2, .Order3	The sorting order to use when sorting .FieldNum, .FieldNum2, and .FieldNum3: 0 (zero) Ascending 1 Descending
.Separator	The type of separator (irrelevant for table rows): 0 (zero) Comma 1 Tab 2 Other (the character that appears in the Other box in the Sort Options dialog box)
.SortColumn	If 1, sorts only the selected column (requires a column selection).
.CaseSensitive	If 1, sorts with case sensitivity.

For more information on the options described above, choose the Help button in the Sort dialog box (Table menu).

See also

Tables Statements and Functions

TableSortAToZ

TableSortZToA

TableSortAToZ

TableSortAToZ

Sorts the paragraphs or table rows in the active document in ascending alphanumeric order. If the first paragraph or table row is a valid header record, it is not included in the sort. **TableSortAToZ** is intended for sorting mail-merge data sources.

_____

See also

Tables Statements and Functions

MailMergeEditDataSource

TableSort

TableSortZToA

TableSortZToA

TableSortZToA

Sorts selected paragraphs or table rows in descending alphanumeric order. If the first paragraph or table row is a valid header record, it is not included in the sort. **TableSortZToA** is intended for sorting mail-merge data sources.

_____

See also

Tables Statements and Functions

MailMergeEditDataSource

TableSort

TableSortAToZ

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableSplit Example

This example splits a table above the fourth row if the table contains more than three rows:

```
If SelInfo(15) > 3 Then
    StartOfColumn
    For i = 1 To 3
        TableSelectRow
        LineDown
    Next
    TableSplit
End If
```

TableSplit

¬Example

TableSplit

Inserts an empty paragraph above the current row in the table. In addition to splitting a table, you can use **TableSplit** to insert an empty paragraph above a table when the table is the first object in the document. If the insertion point or selection is not in a table, an error occurs.

See also

Tables Statements and Functions

TableSplitCells

TableSplitCells

TableSplitCells [**.NumColumns =** *text*]

Splits each selected table cell. Word generates an error if the selection or insertion point is not within a table.

Argument	Explanation
.NumColumns	Specifies the number of cells to split each selected table cell into. If omitted, Word splits each cell into two cells.

See also

Tables Statements and Functions

TableMergeCells

TableSplit

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

TableToText Example

This example converts the selected cells to a tabbed table:

```
TableToText
```

TableToText

¬Example

TableToText [**.ConvertTo** = *number*]

Converts the selected rows to normal text. All the cells in the rows you want to convert must be selected; otherwise, an error occurs. The argument for the **TableToText** statement corresponds to the options in the Convert Table To Text dialog box (Table menu).

Argument	Explanation
.ConvertTo	Determines the character used to separate the contents of each cell: 0 (zero) Paragraph marks 1 or omitted Tab characters (each row ends with a paragraph mark) 2 Commas (each row ends with a paragraph mark) 3 Other (the character that appears in the Other box in the Convert Table To Text dialog box)

See also

Tables Statements and Functions

TableInsertTable

TextToTable

TableUpdateAutoFormat

TableUpdateAutoFormat

Updates the table containing the insertion point with the characteristics of a predefined table format. For example, if you apply a table format with **TableAutoFormat** and then insert rows and columns, the table may no longer match the predefined look. **TableUpdateAutoFormat** restores the format.

_____

See also

Tables Statements and Functions

TableAutoFormat

TextToTable

TextToTable [**.ConvertFrom** = *number*] [**, .NumColumns** = *number*] [**, .NumRows** = *number*] [**, .InitialColWidth** = *number or text*] [**, .Format** = *number*] [**, .Apply** = *number*]

Converts the selected text to a table based on the separator character you specify. If you run **TextToTable** without any arguments, Word uses tab characters as the separator if there are no commas, or commas if there are no tab characters, or paragraph marks if there are no tab characters or commas, or if there is a mixture of tab characters and commas. The arguments for the **TextToTable** statement correspond to the options in the Convert Text To Table dialog box (Table menu).

Argument	Explanation
.ConvertFrom	Specifies the character that separates text elements: 0 (zero) Paragraph marks 1 Tab characters 2 Commas 3 Other (the character that appears in the Other box in the Convert Text To Table dialog box)
.NumColumns	The number of columns to create.
.NumRows	The number of rows to create.
.InitialColWidth	The width of the columns in points or a text measurement. If omitted, column width is calculated so that the table stretches from margin to margin.
.Format	One of the predefined formats listed in the Table AutoFormat dialog box (Table menu): 0 (zero) corresponds to the first format listed in the Formats box ("none"), 1 corresponds to the second format, and so on.
.Apply	Specifies which attributes of the format specified by .Format to apply to the table. For a list of attributes and their values, see TableInsertTable.

See also

Tables Statements and Functions

TableAutoFormat

TableInsertTable

TableToText

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsAdvancedSettings Example

This example sets the date format in the [Microsoft Word] section of WINWORD6.INI:

```
ToolsAdvancedSettings .Application = "Microsoft Word", \
        .Option = "DateFormat", .Setting = "M/D/YYYY", .Set
```

ToolsAdvancedSettings

¬Example

ToolsAdvancedSettings .Application = *text***, .Option =** *text* **, .Setting =** *text* [**, .Delete**] [**, .Set**]

Changes keys in a settings file, such as WINWORD6.INI, WIN.INI, or a private settings file. The arguments for **ToolsAdvancedSettings** correspond to the options in the Advanced Settings dialog box, which you can display by choosing Macro from the Tools menu and running the Word command **ToolsAdvancedSettings**.

Argument	Explanation
.Application	The name of an .INI file section, as listed in the Categories box. Note that by following the section name with a space and the name of an .INI file in parentheses, you can set an key in any settings file. For example, "Macro Settings (MY.INI)" specifies the [Macro Settings] section of the MY.INI file in the Windows directory.
.Option	The key to set.
.Setting	The new setting.
.Delete	Deletes the key.
.Set	Sets the key.

For more information about changing settings in a settings file, see SetPrivateProfileString.

See also

Tools Statements and Functions

GetPrivateProfileString$()

GetProfileString$()

SetPrivateProfileString

SetProfileString

ToolsOptionsFileLocations

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsCalculate Example

In this example, the **ToolsCalculate()** function adds values from two table cells and returns the result in the variable total. The bookmarks "sales" and "other" identify the tables containing the values.

```
total = ToolsCalculate("sum(sales a5, other b5)")
```

ToolsCalculate, ToolsCalculate()

¬Example

ToolsCalculate

ToolsCalculate([*Expression$*]**)**

The **ToolsCalculate** statement evaluates the selection as a mathematical expression, and then displays the result in the status bar. The result is also placed on the Clipboard.

If *Expression$* is not specified, the **ToolsCalculate()** function acts just like the statement except that it returns the result instead of displaying it in the status bar and placing it on the Clipboard. If you do specify *Expression$*, it is evaluated in the same manner as an = (Formula) field. Values in *Expression$* can be table cell references.

For more information on valid expressions, see = (Formula) field.

See also

Tools Statements and Functions

TableAutoSum

TableFormula

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsCompareVersions Example

This example compares the active document to C:\DRAFT\REV1DOC.DOC:

```
ToolsCompareVersions .Name = "C:\DRAFT\REV1DOC.DOC"
```

ToolsCompareVersions

¬Example

ToolsCompareVersions .Name = *text*

Displays revision marks indicating where the active document differs from the specified document. **ToolsCompareVersions** corresponds to the Compare Versions dialog box (Revisions command, Tools menu).

Argument	Explanation
.Name	The name of the document to which the active document is compared; include a path if the document is not in the current directory.

See also

Tools Statements and Functions

ToolsOptionsRevisions

ToolsRevisions

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsCreateEnvelope Example

This example adds an envelope to the active document with addresses specified in the variables to$ and from$. The recipient's address is bold.

```
rtn$ = Chr$(13)
to$ = "Jane Doe"  + rtn$ + "123 Skye St." + \
        rtn$ + "OurTown, WA 98107"
from$ = "John Doe" + rtn$ + "456 Erde Lane" + \
        rtn$ + "OurTown, WA 98107"
FormatAddrFonts .Bold = 1
ToolsCreateEnvelope .EnvAddress = to$, .EnvReturn = from$, \
        .AddToDocument
```

ToolsCreateEnvelope

¬Example

ToolsCreateEnvelope [**.ExtractAddress** = *number*] [**, .EnvAddress** = *text*] [**, .EnvOmitReturn** = *number*] [**, .EnvReturn** = *text*] [**, .PrintBarCode** = *number*] [**, .EnvWidth** = *number or text*] [**, .EnvHeight** = *number or text*] [**, .EnvPaperSize** = *number*] [**, .PrintFIMA** = *number*] [**, .LabelAutoText** = *text*] [**, .UseEnvFeeder** = *number*] [**, .AddrFromLeft** = *number or text*] [**, .AddrFromTop** = *number or text*] [**, .RetAddrFromLeft** = *number or text*] [**, .RetAddrFromTop** = *number or text*] [**, .PrintEnvLabel**] [**, .AddToDocument**]

Creates an envelope that is printed with the active document. The arguments for the **ToolsCreateEnvelope** statement correspond to the options on the Envelopes tab in the Envelopes And Labels dialog box (Tools menu). To specify address formatting, use **FormatAddrFonts** and **FormatRetAddrFonts**.

Argument	Explanation
.ExtractAddress	Specifies whether or not to use the text marked by the "EnvelopeAddress" bookmark (a user-defined bookmark) as the recipient's address: 0 (zero) The "EnvelopeAddress" bookmark is not used. 1 The "EnvelopeAddress" bookmark is used.
.EnvAddress	Text specifying the recipient's address (ignored if .ExtractAddress is specified).
.EnvOmitReturn	Specifies whether or not to omit the return address: 0 The return address is not omitted. 1 The return address is omitted.
.EnvReturn	Text specifying the return address.
.PrintBarCode	Specifies whether or not to add a POSTNET bar code: 1 adds the bar code, 0 (zero) does not. For U.S. mail only.
.EnvWidth, .EnvHeight	The width and height of the envelope, in the current default unit or a text measurement, when .EnvPaperSize corresponds to Custom Size.
.EnvPaperSize	Corresponds to a size in the Envelope Size box in the Envelope Options dialog box: 0 (zero) corresponds to the first size, 1 to the second size, and so on.
.PrintFIMA	Specifies whether or not to add a Facing Identification Mark (FIM A) for use in presorting courtesy reply mail: 1 adds the FIM A, 0 (zero) does not. For U.S. mail only.
.LabelAutoText	The name of an AutoText entry in which the address fields for a data

	source are stored (only recorded when creating envelopes in a mail merge).
.UseEnvFeeder	A number corresponding to the envelope source listed in the Feed From box: 0 (zero) corresponds to the first source, 1 to the second source, and so on.
.AddrFromLeft	The distance, in the current default unit or a text measurement, between the left edge of the envelope and the recipient's address.
.AddrFromTop	The distance, in the current default unit or a text measurement, between the top edge of the envelope and the recipient's address.
.RetAddrFromLeft	The distance, in the current default unit or a text measurement, between the left edge of the envelope and the return address.
.RetAddrFromTop	The distance, in the current default unit or a text measurement, between the top edge of the envelope and the return address.
.PrintEnvLabel	Prints the envelope.
.AddToDocument	Adds a section with the recipient's address and the return address to the beginning of the document.

See also

Tools Statements and Functions

FormatAddrFonts

FormatRetAddrFonts

ToolsCreateLabels

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsCreateLabels Example

This example creates a new document with label text arranged for printing onto a sheet of Avery labels. The argument .LabelListIndex = 0 specifies Avery Standard as the label product, and .LabelIndex = 4 specifies product number 5160 (for addresses). The label text is retrieved from the AutoText entry "MyLabelAddress."

```
ToolsCreateLabels .LabelListIndex = 0, .LabelIndex = 4, \
    .LabelDotMatrix = 0, .LabelTray = 1, .AddToDocument, \
    .LabelAutoText = "MyLabelAddress"
```

ToolsCreateLabels

¬Example

ToolsCreateLabels [**.ExtractAddress** = *number*] [**.LabelListIndex** = *number*] [**, .LabelIndex** = *number*] [**, .LabelDotMatrix** = *number*] [**, .LabelTray** = *number*] [**, .LabelAcross** = *number*] [**, .LabelDown** = *number*] [**, .SingleLabel** = *number*] [**, .LabelRow** = *number*] [**, .LabelColumn** = *number*] [**, .LabelAutoText** = *text*] [**, .LabelText** = *text*] [**, .PrintEnvLabel**] [**, .AddToDocument**] [**, .LabelTopMargin** = *number or text*] [**, .LabelSideMargin** = *number or text*] [**, .LabelVertPitch** = *number or text*] [**, .LabelHorPitch** = *number or text*] [**, .LabelHeight** = *number or text*] [**, .LabelWidth** = *number or text*]

Creates a label or a sheet of labels that is printed with the active document. The arguments for the **ToolsCreateLabels** statement correspond to the options on the Labels tab in the Envelopes And Labels dialog box (Tools menu).

Argument	Explanation
.ExtractAddress	Specifies whether or not to use the text marked by the "EnvelopeAddress" bookmark (a user-defined bookmark) as the label text: 0 (zero) The "EnvelopeAddress" bookmark is not used. 1 The "EnvelopeAddress" bookmark is used.
.LabelListIndex	Corresponds to an item in the Label Products list in the Label Options dialog box (Options button, Envelopes And Labels dialog box). For dot matrix printers, the values are: 1 Avery Standard 3 Avery Intl (UK) 4 Avery Intl (France) 6 CoStar LabelWriter For laser printers, the values are: 0 (zero) Avery Standard 2 Avery Pan European 5 Other
.LabelIndex	Corresponds to an item in the Product Number box in the Label Options dialog box: 0 (zero) corresponds to the first item, 1 to the second item, and so on.
.LabelDotMatrix	Specifies the kind of printer: 0 (zero) Laser 1 Dot matrix
.LabelTray	Corresponds to a tray listed in the Tray list box in the Label Options dialog box: 0 (zero) corresponds to the first tray in the list, 1 to the second, and so on. Available only if .LabelDotMatrix is set to 0 (zero).

.SingleLabel	If 1, allows you to print a single label on a page of labels by specifying .LabelRow and .LabelColumn.
.LabelRow	If .SingleLabel is set to 1, the row containing the label you want to print.
.LabelColumn	If .SingleLabel is set to 1, the column containing the label you want to print.
.LabelAutoText	The name of an AutoText entry in which the label text is stored (ignored if .ExtractAddress is specified).
.LabelText	The text to print on the labels (ignored if .ExtractAddress or .LabelAutoText is specified).
.PrintEnvLabel	Prints the labels.
.AddToDocument	Creates a new document with label text ready for printing.

The following arguments are for creating custom labels and need not be specified (and are ignored) if the combination of .LabelListIndex, .LabelIndex, and .LabelDotMatrix specifies valid labels.

Argument	Explanation
.LabelAcross	The number of labels in a row
.LabelDown	The number of labels in a column
.LabelTopMargin	The width of the top margin on the page of labels, in the current default unit or a text measurement
.LabelSideMargin	The width of the side margins on the page of labels, in the current default unit or a text measurement
.LabelVertPitch	The distance from the top of one label to the top of the next label down, in the current default unit or a text measurement
.LabelHorPitch	The distance from the left edge of one label to the left edge of the next label across, in the current default unit or a text measurement
.LabelHeight	The height of the labels, in the current default unit or a text measurement.
.LabelWidth	The width of the labels, in the current default unit or a text measurement.

See also

Tools Statements and Functions

ToolsCreateEnvelope

ToolsCustomize

ToolsCustomize [**.Tab =** *number*]

Displays the Customize dialog box (Tools menu) with the specified tab selected.

Argument	Explanation
.Tab	The tab to select: 0 (zero) Toolbars 1 Menus 2 Keyboard

See also

Tools Statements and Functions

AddButton

ToolsCustomizeKeyboard

ToolsCustomizeMenuBar

ToolsCustomizeMenus

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsHyphenation Example

This example begins hyphenation, allowing hyphenation of words in all capital letters and setting a hyphenation zone of 24 points:

```
ToolsHyphenation .AutoHyphenation = 1, .HyphenateCaps = 1, \
        .HyphenationZone = "24 pt"
```

ToolsHyphenation

¬Example

ToolsHyphenation [**.AutoHyphenation =** *number*] [**.HyphenateCaps =** *number*] [**, .HyphenationZone =** *number or text*] [**, .LimitConsecutiveHyphens =** *number*]

Hyphenates the selected text or the entire document. The arguments for the **ToolsHyphenation** statement correspond to the options in the Hyphenation dialog box (Tools menu).

Argument	Explanation
.AutoHyphenation	If 1, hyphenates without prompting for verification.
.HyphenateCaps	If 1, allows hyphenation of words in all capital letters.
.HyphenationZone	A measurement for the hyphenation zone in points or a text measurement. The hyphenation zone is the maximum amount of space to allow between the end of a line and the right margin.
.LimitConsecutiveHyphens	The maximum number of consecutive lines that can end with hyphens (or 0 (zero) for No Limit).

See also

Tools Statements and Functions

ToolsHyphenationManual

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsHyphenationManual Example

This example sets a hyphenation zone of 16 points before running **ToolsHyphenationManual**. An error handler prevents error 102 ("Command failed") from appearing if the user cancels the Manual Hyphenation dialog box.

```
ToolsHyphenation .AutoHyphenation = 0, .HyphenationZone = 16
On Error Goto trap
ToolsHyphenationManual
trap:
If Err = 102 Then Goto bye Else Error Err
bye:
```

ToolsHyphenationManual

¬Example

ToolsHyphenationManual

Hyphenates the document, displaying the Manual Hyphenation dialog box (Manual button, Hyphenation dialog box, Tools menu) each time Word proposes that a word be hyphenated so the user can manually adjust the placement of hyphens. To set hyphenation options before running **ToolsHyphenationManual**, use the **ToolsHyphenation** statement and set .AutoHyphenation to 0 (zero).

_____

See also

Tools Statements and Functions

ToolsHyphenation

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsLanguage Example

This example makes Brazilian Portuguese an attribute of the Normal style:

```
ToolsLanguage .Language = "Português (BR)", .Default
```

ToolsLanguage

¬Example

ToolsLanguage .Language = *text* [**, .Default**]

Sets the language format for the selected text. The language format identifies the text for the proofing tools in Word so that rules appropriate to the specified language are used.

Argument	Explanation
.Language	The name of the language
.Default	Adds the specified language format to the definition of the active template's Normal style

The following list includes all the valid settings for .Language.

To specify this language	Set .Language to this text
No Proofing	0 (zero)
	Do not include the text "(zero)"
Brazilian Portuguese	Português (BR)
Catalan	Català
Danish	Dansk
Dutch	Nederlands
Dutch (preferred)	Nederlands (voorkeur)
English (Australian)	English (AUS)
English (U.K.)	English (UK)
English (U.S.)	English (US)
Finnish	Suomi
French	Français
French Canadian	Canadien Français
German	Deutsch
Italian	Italiano
Norwegian Bokmål	Norsk Bokmål
Norwegian Nynorsk	Norsk Nynorsk
Portuguese	Português (POR)
Spanish	Español
Swedish	Svenska
Swiss German	Deutsch (Schweiz)

See also

Tools Statements and Functions

CountLanguages()

FormatDefineStyleLang

Language

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsMergeRevisions Example

This example opens REVISE1.DOC, merges revision marks into ORIGINAL.DOC, and then performs the same action with REVISE2.DOC:

```
FileOpen .Name = "C:\DOCS\REVISE1.DOC"
ToolsMergeRevisions .Name = "C:\DOCS\ORIGINAL.DOC"
FileOpen .Name = "C:\DOCS\REVISE2.DOC"
ToolsMergeRevisions .Name = "C:\DOCS\ORIGINAL.DOC"
```

ToolsMergeRevisions

¬Example

ToolsMergeRevisions .Name = *text*

Merges the revision marks from the active document into the specified document, which Word opens if it is not open already. If the active document minus revision marks is not equivalent to the specified document minus revision marks, an error occurs. You can use a series of **FileOpen** instructions followed by **ToolsMergeRevisions** instructions to collect revision marks from multiple review copies into one document.

Argument	Explanation
.Name	The path and filename of the document from which you want to merge revisions

See also

Tools Statements and Functions

ToolsCompareVersions

ToolsOptionsRevisions

ToolsReviewRevisions

ToolsRevisions

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptions Example

This example displays the View tab in the Options dialog box. The On Error Resume Next instruction prevents Word from displaying an error message if the user chooses Cancel.

```
On Error Resume Next
ToolsOptions .Tab = 0
```

ToolsOptions

¬Example

ToolsOptions .Tab = *number*

Displays the Options dialog box (Tools menu) with the specified tab selected. Unless you precede **ToolsOptions** with an **On Error** instruction, Word displays an error message if the dialog box is canceled.

Argument	Explanation
.Tab	The tab to select: 0 (zero) View 1 General 2 Edit 3 Print 4 Save 5 Spelling 6 Grammar 7 AutoFormat 8 Revisions 9 User Info 10 Compatibility 11 File Locations

See also

Tools Statements and Functions

ToolsCustomize

ToolsOptionsAutoFormat

ToolsOptionsAutoFormat [**.PreserveStyles** = *number*] [**, .ApplyStylesHeadings** = *number*] [**, .ApplyStylesLists** = *number*] [**, .ApplyStylesOtherParas** = *number*] [**, .AdjustParaMarks** = *number*] [**, .AdjustTabsSpaces** = *number*] [**, .AdjustEmptyParas** = *number*] [**, .ReplaceQuotes** = *number*] [**, .ReplaceSymbols** = *number*] [**, .ReplaceBullets** = *number*]

Sets automatic formatting options for **FormatAutoFormat** and **FormatAutoFormatBegin**. The arguments for the **ToolsOptionsAutoFormat** statement correspond to the options on the AutoFormat tab in the Options dialog box (Tools menu).

Argument	Explanation
.PreserveStyles	If 1, preserves previously applied styles.
.ApplyStylesHeadings	If 1, applies automatic styles to headings.
.ApplyStylesLists	If 1, applies automatic styles to lists.
.ApplyStylesOtherParas	If 1, applies automatic styles to paragraphs.
.AdjustParaMarks	If 1, adds or removes paragraph marks in a document (for example, removes extra paragraph marks from lines of text copied from an electronic mail message).
.AdjustTabsSpaces	If 1, adjusts tab settings and spaces automatically.
.AdjustEmptyParas	If 1, removes empty paragraph marks used to add space between paragraphs.
.ReplaceQuotes	If 1, replaces straight double quotation marks, single quotation marks, and apostrophes with curled, or "smart," quotation marks and apostrophes. "Xxx" “Xxx”
.ReplaceSymbols	If 1, replaces "(r)", "(c)", and "(tm)" with ®, ©, and ™ symbols from the Symbol dialog box (Insert menu).
.ReplaceBullets	If 1, replaces bullet characters with bullets from the Bullets And Numbering dialog box (Format menu).

For more information on the options described above, select the AutoFormat tab in the Options dialog box, and then choose the Help button.

See also

Tools Statements and Functions

ToolsAutoCorrect

ToolsOptionsCompatibility

ToolsOptionsCompatibility [**.Product** = *text*] [**, .Default**] [**, .NoTabHangIndent** = *number*] [**, .NoSpaceRaiseLower** = *number*] [**, .PrintColBlack** = *number*] [**, .WrapTrailSpaces** = *number*] [**, .NoColumnBalance** = *number*] [**, .ConvMailMergeEsc** = *number*] [**, .SuppressSpBfAfterPgBrk** = *number*] [**, .SuppressTopSpacing** = *number*] [**, .OrigWordTableRules** = *number*] [**, .TransparentMetafiles** = *number*] [**, .ShowBreaksInFrames** = *number*] [**, .SwapBordersFacingPages** = *number*]

Adjusts the display of certain elements in the active document to mimic the display in other word-processing applications. For example, you can specify that spaces at the end of a line wrap to the next line as they do in WordPerfect. The arguments for the **ToolsOptionsCompatibility** statement correspond to the options on the Compatibility tab in the Options dialog box (Tools menu).

Argument	Explanation
.Product	The name of the product whose display you want to mimic, as listed in the Recommended Options For box. If you specify .Product, display options are determined automatically.
.Default	Makes the display options you specify the default for new documents based on the active template.
.NoTabHangIndent	If 1, a tab stop is not added automatically to a paragraph formatted with a hanging indent.
.NoSpaceRaiseLower	If 1, extra line spacing is not added for raised and lowered characters.
.PrintColBlack	If 1, colors are printed as black on printers that don't support color.
.WrapTrailSpaces	If 1, spaces at the end of lines wrap to the next line.
.NoColumnBalance	If 1, text columns are not balanced above continuous section breaks.
.ConvMailMergeEsc	If 1, correctly interprets characters preceded by backslashes (\) in Word version 2.*x* mail-merge data sources. For example, \" is interpreted as ".
.SuppressSpBfAfterPgBrk	If 1, space before or after hard page and column breaks is removed.
.SuppressTopSpacing	If 1, extra line spacing at the top of the page is removed.
.OrigWordTableRules	If 1, table borders are combined as in Word for the Macintosh version 5.*x*.
.TransparentMetafiles	If 1, the area behind metafile pictures is not blanked.
.ShowBreaksInFrames	If 1, Word displays manual, or

	"hard," page or column breaks in any frames that contain them.
.SwapBordersFacingPages	If 1, Word prints a left paragraph border (not a box) on the right on odd numbered pages if the Different Odd And Even Headers or Mirror Margins check box (Page Setup dialog box, File menu) is selected.

Note that if you specify both .Product and at least one other display option, the display option settings take precedence.

For more information on the options described above, select the Compatibility tab in the Options dialog box, and then choose the Help button.

_____

See also

Tools Statements and Functions

FontSubstitution

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsEdit Example

This example toggles the Drag-And-Drop Text Editing check box:

```
Dim dlg As ToolsOptionsEdit
GetCurValues dlg
If dlg.DragAndDrop Then
    dlg.DragAndDrop = 0
Else
    dlg.DragAndDrop = 1
End If
ToolsOptionsEdit dlg
```

ToolsOptionsEdit

¬Example

ToolsOptionsEdit [**.ReplaceSelection** = *number*] [**, .DragAndDrop** = *number*] [**, .AutoWordSelection** = *number*] [**, .InsForPaste** = *number*] [**, .Overtype** = *number*] [**, .SmartCutPaste** = *number*] [**, .AllowAccentedUppercase** = *number*] [**, .PictureEditor** = *text*]

Sets editing options. The arguments for the **ToolsOptionsEdit** statement correspond to the options on the Edit tab in the Options dialog box (Tools menu).

Argument	Explanation
.ReplaceSelection	If 1, replaces selected text with typed text.
.DragAndDrop	If 1, allows drag-and-drop editing.
.AutoWordSelection	If 1, dragging selects one word at a time instead of one character at a time.
.InsForPaste	If 1, allows the INS key to be used for pasting the Clipboard contents.
.Overtype	If 1, replaces text following the insertion point with typed text.
.SmartCutPaste	If 1, automatically adjusts spacing between words and punctuation when cutting and pasting.
.AllowAccentedUppercase	If 1, allows proofing tools and the Change Case feature to suggest that Word add an accent mark to an uppercase letter.
.PictureEditor	The name of the application you want to use to edit pictures, as it appears in the Picture Editor box.

For more information on the options described above, select the Edit tab in the Options dialog box, and then choose the Help button.

See also

Tools Statements and Functions

Overtype

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsGeneral Example

Before opening C:\TMP\TEST.DOC, this example ensures that no filename will be "bumped off" the list of recently used files (unless the list already contains the maximum of nine filenames):

```
Dim dlg As ToolsOptionsGeneral
GetCurValues dlg
num = CountFiles()
If num < 9 Then
    num$ = LTrim$(Str$(num))
    If dlg.RecentFileCount = num$ Then
        dlg.RecentFileCount = num + 1
        ToolsOptionsGeneral dlg
    End If
End If
FileOpen .Name = "C:\TMP\TEST.DOC"
```

ToolsOptionsGeneral

¬Example

ToolsOptionsGeneral [**.Pagination** = *number*] [**, .WPHelp** = *number*] [**, .WPDocNavKeys** = *number*] [**, .BlueScreen** = *number*] [**, .ErrorBeeps** = *number*] [**, .Effects3d** = *number*] [**, .UpdateLinks** = *number*] [**, .SendMailAttach** = *number*] [**, .RecentFiles** = *number*] [**, .RecentFileCount** = *number*] [**, .Units** = *number*] [**, .ButtonFieldClicks** = *number*]

Sets general options. The arguments for the **ToolsOptionsGeneral** statement correspond to the options on the General tab in the Options dialog box (Tools menu).

Argument	Explanation
.Pagination	If 1, allows background repagination.
.WPHelp	If 1, enables WordPerfect Help.
.WPDocNavKeys	If 1, enables WordPerfect document navigation keys.
.BlueScreen	If 1, changes the document window to display white text on a blue background.
.ErrorBeeps	If 1, Word beeps when an action occurs that generates an error.
.Effects3d	If 1, Word displays dialog boxes with three-dimensional effects.
.UpdateLinks	If 1, automatically updates linked information when you open documents.
.SendMailAttach	If 1, the Send command (File menu) sends the active document as an attachment instead of text in an electronic mail message.
.RecentFiles	If 1, lists recently used files above the Exit command on the File menu.
.RecentFileCount	A number from 1 to 9 corresponding to the maximum number of recently used files you want to be listed.
.Units	Sets the default unit of measurement: 0 (zero) Inches 1 Centimeters 2 Points 3 Picas
.ButtonFieldClicks	Sets the number of clicks (1 or 2) required to run a macro with a MACROBUTTON field.

For more information on the options described above, select the General tab in the Options dialog box, and then choose the Help button.

See also

Tools Statements and Functions

Beep

CountFiles()

HelpWordPerfectHelpOptions

ToolsRepaginate

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsPrint Example

This example specifies that documents be printed in reverse page order:

```
ToolsOptionsPrint .Reverse = 1
```

ToolsOptionsPrint

¬Example

ToolsOptionsPrint [**.Draft** = *number*] [**, .Reverse** = *number*] [**, .UpdateFields** = *number*] [**, .Summary** = *number*] [**, .ShowCodes** = *number*] [**, .Annotations** = *number*] [**, .ShowHidden** = *number*] [**, .EnvFeederInstalled** = *number*] [**, .UpdateLinks** = *number*] [**, .Background** = *number*] [**, .DrawingObjects** = *number*] [**, .FormsData** = *number*] [**, .DefaultTray** = *text*]

Sets options for printing a document. The arguments for the **ToolsOptionsPrint** statement correspond to the options on the Print tab in the Options dialog box (Tools menu).

Argument	Explanation
.Draft	If 1, prints in draft output.
.Reverse	If 1, prints pages in reverse order.
.UpdateFields	If 1, updates all fields in the document when printed.
.Summary	If 1, prints summary information with the document.
.ShowCodes	If 1, prints field codes instead of field results.
.Annotations	If 1, prints annotations with the document.
.ShowHidden	If 1, prints all hidden text with the document.
.EnvFeederInstalled	If 1, indicates that an envelope feeder is installed.
.UpdateLinks	If 1, automatically updates all linked information when you print documents.
.Background	If 1, allows background printing.
.DrawingObjects	If 1, prints Word drawing objects with the document.
.FormsData	If 1, prints only the data entered by the user in an online form onto a preprinted form.
.DefaultTray	Specifies the default paper tray as it appears in the Default Tray box.

For more information on the options described above, select the Print tab in the Options dialog box, and then choose the Help button.

See also

Tools Statements and Functions

FilePrint

FilePrintSetup

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsRevisions Example

When revision marking is active, this example instructs Word to format inserted text with a double underline and to add blue revision lines in the left margin:

```
ToolsOptionsRevisions .InsertedTextMark = 4, \
        .RevisedLinesMark = 1, .RevisedLinesColor = 2
```

ToolsOptionsRevisions

¬Example

ToolsOptionsRevisions [**.InsertedTextMark =** *number*] [**, .InsertedTextColor =** *number*] [**, .DeletedTextMark =** *number*] [**, .DeletedTextColor =** *number*] [**, .RevisedLinesMark =** *number*] [**, .RevisedLinesColor =** *number*]

Sets options for marking revisions. The arguments for the **ToolsOptionsRevisions** statement correspond to the options on the Revisions tab in the Options dialog box (Tools menu). The options affect documents in which revision marking is active, and documents you compare to other documents with the Compare Versions command (Revisions dialog box, Tools menu).

Argument	Explanation
.InsertedTextMark	The format for inserted text: 0 (zero) None 1 Bold 2 Italic 3 Underline 4 Double underline
.DeletedTextMark	The format for deleted text: 0 (zero) Hidden 1 Strikethrough
.RevisedLinesMark	The position for revision lines: 0 (zero) None 1 Left border 2 Right border 3 Outside border
.InsertedTextColor, .DeletedTextColor, .RevisedLinesColor	The colors for inserted text, deleted text, and revision lines. If omitted, the color is set to "Auto." For a list of values, see CharColor.

For more information on the options described above, select the Revisions tab in the Options dialog box, and then choose the Help button.

See also

Tools Statements and Functions

ToolsCompareVersions

ToolsRevisions

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsSave Example

This example sets an interval of 10 minutes between automatic saves:

```
ToolsOptionsSave .AutoSave = 1, .SaveInterval = "10"
```

ToolsOptionsSave

¬Example

ToolsOptionsSave [**.CreateBackup** = *number*] [**, .FastSaves** = *number*] [**, .SummaryPrompt** = *number*] [**, .GlobalDotPrompt** = *number*] [**, .NativePictureFormat** = *number*] [**, .EmbedFonts** = *number*] [**, .FormsData** = *number*] [**, .AutoSave** = *number*] [**, .SaveInterval** = *text*] [**, .Password** = *text*] [**, .WritePassword** = *text*] [**, .RecommendReadOnly** = *number*]

Sets options for saving documents. The arguments for the **ToolsOptionsSave** statement correspond to the options on the Save tab in the Options dialog box (Tools menu).

Argument	Explanation
.CreateBackup	If 1, creates a backup copy every time you save, and fast saves are not allowed.
.FastSaves	If 1, allows fast saves.
.SummaryPrompt	If 1, prompts for summary information when you save a new document.
.GlobalDotPrompt	If 1 and you make changes to the Normal template, displays a message box asking if you want save the changes when you quit Word; if 0 (zero), changes to the Normal template are saved automatically.
.NativePictureFormat	If 1, saves only the Windows version of imported graphics.
.EmbedFonts	If 1, embeds TrueType fonts when you save.
.FormsData	If 1, saves the data entered by a user in a form as a tab-delimited record for use in a database.
.AutoSave	If 1, allows automatic saving.
.SaveInterval	Specifies the time interval for saving documents automatically, in minutes; available only if .AutoSave has been set to 1.
.Password	The password for opening the document.
.WritePassword	The password for saving changes to the document.
.RecommendReadOnly	If 1, displays a message box upon opening the document suggesting that it be opened as read-only.

For more information on the options described above, select the Save tab in the Options dialog box, and then choose the Help button.

See also

Tools Statements and Functions

FileSave

FileSaveAll
FileSaveAs

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsUserInfo Example

This example sets the current user name and initials:

```
ToolsOptionsUserInfo .Name = "Gina Fiori", .Initials = "GF"
```

ToolsOptionsUserInfo

¬Example

ToolsOptionsUserInfo [**.Name** = *text*] [**, .Initials =** *text*] [**, .Address =** *text*]

Changes user information. The arguments for the **ToolsOptionsUserInfo** statement correspond to the options on the User Info tab in the Options dialog box (Tools menu).

Argument	Explanation
.Name	The name of the current user
.Initials	The initials of the current user
.Address	The mailing address of the current user

For more information on the options described above, select the User Info tab in the Options dialog box, and then choose the Help button.

See also

Tools Statements and Functions

DocumentStatistics

FileSummaryInfo

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsRepaginate Example

This example repaginates the document if it has been changed since the last time it was saved:

```
If IsDocumentDirty() Then ToolsRepaginate
```

ToolsRepaginate

¬Example

ToolsRepaginate

Forces repagination of the entire document.

_____

See also

Tools Statements and Functions

ToolsOptionsGeneral

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsReviewRevisions Example

This example displays revision marks, selects the next marked text, and then accepts the revisions:

```
ToolsReviewRevisions .ShowMarks
ToolsReviewRevisions .FindNext
ToolsReviewRevisions .AcceptRevisions
```

ToolsReviewRevisions

¬Example

ToolsReviewRevisions [**.ShowMarks**] [**, .HideMarks**] [**, .Wrap =** *number*] [**, .FindPrevious**] [**, .FindNext**] [**, .AcceptRevisions**] [**, .RejectRevisions**]

Searches for revision marks or accepts or rejects the selected revisions. The arguments for **ToolsReviewRevisions** correspond to the options in the Review Revisions dialog box (Revisions command, Tools menu).

You can use only one of the following arguments per **ToolsReviewRevisions** instruction.

Argument	Explanation
.ShowMarks	Displays revision marks in the active document.
.HideMarks	Hides revision marks in the active document.
.Wrap	Controls what happens if the search begins at a point other than the beginning of the document and the end of the document is reached: 0 (zero) or omitted — The search operation ends and the macro continues. 1 — The search continues from the beginning of the document to the point where the search began. 2 — Word displays a message box asking whether to continue the search from the beginning of the document.
.FindPrevious	Searches toward the beginning of the document and selects the nearest text with revision marks.
.FindNext	Searches toward the end of the document and selects the nearest text with revision marks.
.AcceptRevisions	Accepts the revisions to the selected text.
.RejectRevisions	Cancels the revisions to the selected text.

See also

Tools Statements and Functions

ToolsCompareVersions

ToolsOptionsRevisions

ToolsRevisions

ToolsRevisionType()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsRevisionAuthor$() Example

This example finds the next revision and checks the revision author. If the revision author is Stella Richards, the revision is accepted.

```
ToolsRevisions .ViewRevisions = 1
ToolsReviewRevisions .FindNext
If ToolsRevisionAuthor$() = "Stella Richards" Then
    ToolsReviewRevisions .AcceptRevisions
End If
```

ToolsRevisionAuthor$()

¬Example

ToolsRevisionAuthor$()

Returns the name of the person who made the selected revision. If the selection does not include revision marks, **ToolsRevisionAuthor$()** returns an empty string ("").

_____

See also

Tools Statements and Functions

ToolsReviewRevisions

ToolsRevisionDate$()

ToolsRevisions

ToolsRevisionType()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsRevisionDate$() Example

This example uses the **ToolsRevisionType()** function to determine whether the selection contains a revision. If so, a message box displays the date and time the selected revision was made. If the selected text contains no revision, Word displays an appropriate message box.

```
If ToolsRevisionType() <> 0 Then
    MsgBox "Revision made " + ToolsRevisionDate$() + "."
Else
    MsgBox "No revisions are selected."
End If
```

ToolsRevisionDate$()

¬Example

ToolsRevisionDate$()

Returns the date and time the selected revision was made. If the selection does not include revision marks, **ToolsRevisionDate$()** returns an empty string.

_____

See also

Tools Statements and Functions

ToolsReviewRevisions

ToolsRevisionAuthor$()

ToolsRevisionDate()

ToolsRevisions

ToolsRevisionType()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsRevisionDate() Example

This example finds the next revision and checks the revision date. The revision is rejected if it was made more than 10 days before the current date.

```
ToolsRevisions .ViewRevisions = 1
ToolsReviewRevisions .FindNext
If (Now() - ToolsRevisionDate()) > 10 Then
    ToolsReviewRevisions .RejectRevisions
End If
```

ToolsRevisionDate()

¬Example

ToolsRevisionDate()

Returns a serial number corresponding to the date and time the selected revision was made, or -1 if the selection contains no revisions. For information on serial numbers, see DateSerial().

_____

See also

Tools Statements and Functions

ToolsReviewRevisions

ToolsRevisionAuthor$()

ToolsRevisionDate$()

ToolsRevisions

ToolsRevisionType()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsRevisions Example

This example activates revision marking in the active document, but prevents the revision marks from appearing while the document is edited:

```
ToolsRevisions .MarkRevisions = 1, .ViewRevisions = 0
```

ToolsRevisions

¬Example

ToolsRevisions [**.MarkRevisions =** *number*] [**, .ViewRevisions =** *number*] [**, .PrintRevisions =** *number*] [**, .AcceptAll**] [**, .RejectAll**]

Specifies how revisions are marked and reviewed in the active document. The arguments for the **ToolsRevisions** statement correspond to the options in the Revisions dialog box (Tools menu).

Argument	Explanation
.MarkRevisions	If 1, activates revision marking.
.ViewRevisions	If 1, revision marks appear in the document when you edit it with revision marking on.
.PrintRevisions	Specifies whether or not to include revision marks in the printed document: 0 (zero) Revision marks do not appear (revisions are printed as if they were accepted). 1 Revision marks appear.
.AcceptAll	Accepts all revisions in the active document.
.RejectAll	Rejects all revisions in the active document.

See also

Tools Statements and Functions

ToolsCompareVersions

ToolsMergeRevisions

ToolsOptionsRevisions

ToolsReviewRevisions

ToolsRevisionType()

ToolsRevisionType()

ToolsRevisionType()

Returns one of the following values, corresponding to the type of revision made to the selected text.

Value	Explanation
0 (zero)	If the selection contains no revisions
1	If all or part of the selection contains text marked as inserted
2	If all or part of the selection contains text marked as deleted
3	If the selection contains a replacement (text marked as inserted followed immediately by text marked as deleted)
4	If the selection contains more than one revision

For an example, see the entry for ToolsRevisionDate$() Example.

See also

Tools Statements and Functions

ToolsReviewRevisions

ToolsRevisionAuthor$()

ToolsRevisionDate()

ToolsRevisionDate$()

ToolsRevisions

ToolsShrinkToFit

ToolsShrinkToFit

Attempts to decrease the font size of text just enough to fit the active document on one fewer pages. **ToolsShrinkToFit** is handy for saving paper when printing if the last page contains only a few lines of text. If Word is unable to perform the operation, a message box is displayed.

_____

See also

Tools Statements and Functions

ViewZoomWholePage

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsWordCount Example

This example counts the number of pages, words, characters, paragraphs, and lines in the active document --- including text in footnotes and endnotes --- and then displays the word count in a message box:

```
ToolsWordCount .CountFootnotes = 1
Dim dlg As ToolsWordCount
GetCurValues dlg
MsgBox "Current word count: " + dlg.Words
```

ToolsWordCount

¬Example

ToolsWordCount [**.CountFootnotes** = *number*] [**, .Pages** = *text*] [**, .Words** = *text*] [**, .Characters** = *text*] [**, .Paragraphs** = *text*] [**, .Lines** = *text*]

Counts the number of pages, words, characters, paragraphs, and lines in the active document. The .CountFootnotes argument is the only one that can be set; the remaining arguments are read-only and can be used with a dialog record to return information about the active document. For more information, see the example for this entry. For information about dialog records, see Chapter 4, "Advanced WordBasic," in the Microsoft Word Developer's Kit.

Argument	Explanation
.CountFootnotes	If 1, text in footnotes and endnotes is included in the count.
.Pages	The number of pages in the document.
.Words	The number of words in the document.
.Characters	The number of characters in the document.
.Paragraphs	The number of paragraphs in the document.
.Lines	The number of lines in the document.

See also

Tools Statements and Functions

DocumentStatistics

FileSummaryInfo

ToolsEnvelopesAndLabels

To create or print an envelope using WordBasic, use ToolsCreateEnvelope. To create or print a label or sheet of labels, use ToolsCreateLabels.

ClosePreview

ClosePreview

Returns the active document from print preview to the previous view. **ClosePreview** has no effect if the active document is not displayed in print preview.

See also

View Statements and Functions

FilePrintPreview

CloseViewHeaderFooter

CloseViewHeaderFooter

Hides the Header And Footer toolbar and moves the insertion point to its previous location in the document area. If the insertion point is not in a header or footer, an error occurs.

_____

See also

View Statements and Functions

GoToHeaderFooter

ShowNextHeaderFooter

ShowPrevHeaderFooter

ViewHeader

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ShowAll Example

This example displays all nonprinting characters and then searches for hidden text:

```
ShowAll 1
EditFindClearFormatting
EditFindFont .Hidden = 1
EditFind .Find = "", .Format = 1
```

ShowAll, ShowAll()

¬Example

ShowAll [*On*]

ShowAll()

The **ShowAll** statement displays all nonprinting characters, such as hidden text, tab marks, space marks, and paragraph marks. **ShowAll** corresponds to the All check box on the View tab in the Options dialog box (Tools menu).

Argument	Explanation
On	Specifies whether to hide or display all nonprinting characters: 1 Displays nonprinting characters. 0 (zero) Hides nonprinting characters. Omitted Toggles the display.

If your macro depends on nonprinting characters being visible in the document, it's a good idea to include ShowAll 1 as one of the first instructions.

The **ShowAll()** function returns the following values.

Value	Explanation
0 (zero)	The All check box is selected.
-1	The All check box is cleared.

See also

View Statements and Functions

ToolsOptionsView

ShowNextHeaderFooter

ShowNextHeaderFooter

If the insertion point is in a header, moves to the next header within the current section (for example, from an odd header to an even header) or to the first header in the following section. If the insertion point is in a footer, **ShowNextHeaderFooter** moves to the next footer. If the insertion point is in the last header or footer in the last section of the document, or is not in a header or footer, an error occurs.

See also

View Statements and Functions

CloseViewHeaderFooter

FormatHeaderFooterLink

GoToHeaderFooter

ShowPrevHeaderFooter

ToggleHeaderFooterLink

ViewFooter

ViewHeader

ShowPrevHeaderFooter

ShowPrevHeaderFooter

If the insertion point is in a header, moves to the previous header within the current section (for example, from an even header to an odd header) or to the last header in the previous section. If the insertion point is in a footer, **ShowNextHeaderFooter** moves to the previous footer. If the insertion point is in the first header or footer in the first section of the document, or is not in a header or footer, an error occurs.

See also

View Statements and Functions

CloseViewHeaderFooter

FormatHeaderFooterLink

GoToHeaderFooter

ShowNextHeaderFooter

ToggleHeaderFooterLink

ViewFooter

ViewHeader

ToggleFull

ToggleFull

Toggles full screen mode on and off.

_____

See also

View Statements and Functions

ToolsOptionsView

ViewToolbars

TogglePortrait

TogglePortrait

Switches the selected sections between portrait and landscape page orientations. If the selected sections have different page orientations, an error occurs.

See also

View Statements and Functions

FilePageSetup

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ToolsOptionsView Example

This example toggles the display of hidden text:

```
Dim dlg As ToolsOptionsView
GetCurValues dlg
dlg.ShowAll = 0
If dlg.Hidden Then
    dlg.Hidden = 0
Else
    dlg.Hidden = 1
End If
ToolsOptionsView dlg
```

ToolsOptionsView

¬Example

ToolsOptionsView [**.DraftFont** = *number*] [**, .WrapToWindow** = *number*] [**, .PicturePlaceHolders** = *number*] [**, .FieldCodes** = *number*] [**, .BookMarks** = *number*] [**, .FieldShading** = *number*] [**, .StatusBar** = *number*] [**, .HScroll** = *number*] [**, .VScroll** = *number*] [**, .StyleAreaWidth** = *number or text*] [**, .Tabs** = *number*] [**, .Spaces** = *number*] [**, .Paras** = *number*] [**, .Hyphens** = *number*] [**, .Hidden** = *number*] [**, .ShowAll** = *number*] [**, .Drawings** = *number*] [**, .Anchors** = *number*] [**, .TextBoundaries** = *number*] [**, .VRuler** = *number*]

Displays or hides various elements in the active document or macro-editing window and the Word window. With the exception of .StatusBar, which controls the display of the status bar no matter which window is active, the arguments for **ToolsOptionsView** control the display of elements on a window-by-window basis. The arguments correspond to the options on the View tab in the Options dialog box (Tools menu). As indicated in the following table, not all arguments are available in every view or in the macro-editing window. Specifying an argument that is not available in the active view generates an error.

Argument	Explanation	Available in these views
.DraftFont	If 1, shows all text in the same font without formatting.	Normal, outline, macro
.WrapToWindow	If 1, Word wraps text within document windows; column formatting is ignored.	Normal, outline, macro
.PicturePlaceHolders	If 1, displays placeholders for graphics.	All
.FieldCodes	If 1, displays field codes.	All
.BookMarks	If 1, displays bold brackets around text marked with a bookmark.	All
.FieldShading	Specifies when to display fields with shading: 0 (zero) Never 1 Always 2 When Selected	All
.StatusBar	If 1, displays the status bar.	All
.HScroll	If 1, displays horizontal scroll bars in document windows.	All
.VScroll	If 1, displays vertical scroll bars in document windows.	All
.StyleAreaWidth	Sets the width of the style area in twips (20 twips = 1 point; 72 points = 1 inch) or a text measurement.	Normal, outline
.Tabs	If 1, displays tab marks.	All

.Spaces	If 1, displays space marks.	All
.Paras	If 1, displays paragraph marks.	All
.Hyphens	If 1, displays optional hyphens.	All
.Hidden	If 1, displays hidden text.	All
.ShowAll	If 1, displays all nonprinting characters.	All
.Drawings	If 0 (zero), hides any drawing objects you've created in your documents.	Page layout
.Anchors	If 1, displays anchors next to items that may be positioned.	Page layout
.TextBoundaries	If 1, displays text boundaries.	Page layout
.VRuler	If 1, displays the vertical ruler.	Page layout

For more information on the options described above, select the View tab in the Options dialog box and then choose the Help button.

_____

See also

View Statements and Functions

ShowAll

TableGridlines

ToggleFull

ToolsOptionsCompatibility

ViewDraft

ViewRibbon

ViewRuler

ViewStatusBar

ViewDraft, ViewDraft()

ViewDraft [*On*]

ViewDraft()

The **ViewDraft** statement turns draft mode on or off for the active document or macro-editing window. Draft mode is specified by the Draft Font check box on the View tab in the Options dialog box (Tools menu).

Argument	Explanation
On	Specifies whether to turn draft mode on or off: 0 (zero) Turns draft mode off. 1 Turns draft mode on. Omitted Toggles draft mode.

The **ViewDraft()** function returns the following values.

Value	Explanation
0 (zero)	If draft mode is off
-1	If draft mode is on

See also

View Statements and Functions

ToolsOptionsView

ViewFieldCodes, ViewFieldCodes()

ViewFieldCodes [*On*]

ViewFieldCodes()

The **ViewFieldCodes** statement controls the display of all fields in the active document. The display of field codes is specified by the Field Codes check box on the View tab in the Options dialog box (Tools menu). You can control the display of selected fields with the **ToggleFieldDisplay** statement.

Argument	Explanation
On	Specifies how to display fields: 0 (zero) Displays field results. 1 Displays field codes. Omitted Toggles the display of fields.

The **ViewFieldCodes()** function returns the following values.

Value	Explanation
0 (zero)	If field results are displayed
-1	If field codes are displayed

See also

View Statements and Functions

ToggleFieldDisplay

ToolsOptionsView

ViewFooter, ViewFooter()

ViewFooter

ViewFooter()

The **ViewFooter** statement switches the active document to page layout view, positions the insertion point in the footer area, and then displays the Header And Footer toolbar. If the Header And Footer toolbar is already displayed, **ViewFooter** hides it and moves the insertion point to the document area.

The **ViewFooter()** function returns the following values.

Value	Explanation
0 (zero)	If the insertion point is not in the footer area
-1	If the insertion point is in the footer area

See also

View Statements and Functions

CloseViewHeaderFooter

ViewHeader

ViewFootnoteArea, ViewFootnoteArea()

ViewFootnoteArea [*On*]

ViewFootnoteArea()

In normal and outline views, the **ViewFootnoteArea** statement opens or closes the footnote pane. In page layout view, **ViewFootnoteArea** moves the insertion point to or from the footnote area. If there are no footnotes in the active document, an error occurs.

Argument	Explanation
On	Specifies whether to open or close the footnote pane: 0 (zero) In normal and outline views, closes the footnote pane; in page layout view, has no effect. 1 In normal and outline views, opens the footnote pane; in page layout view, moves the insertion point between an footnote and its associated reference mark in the document text. Omitted In normal and outline views, toggles the display of the footnote pane; in page layout view, moves the insertion point between an footnote and its associated reference mark in the document text.

The **ViewFootnoteArea()** function returns the following values.

Value	Explanation
0 (zero)	If the footnote pane is closed
-1	If the footnote pane is open

See also

View Statements and Functions

ViewAnnotations

ViewEndnoteArea

ViewFootnotes

ViewHeader, ViewHeader()

ViewHeader

ViewHeader()

The **ViewHeader** statement switches the active document to page layout view, positions the insertion point in the header area, and then displays the Header And Footer toolbar. If the Header And Footer toolbar is already displayed, **ViewHeader** hides it and moves the insertion point to the document area.

The **ViewHeader()** function returns the following values.

Value	Explanation
0 (zero)	If the insertion point is not in the header area
-1	If the insertion point is in the header area

See also

View Statements and Functions

ViewFooter

ViewMasterDocument, ViewMasterDocument()

ViewMasterDocument

ViewMasterDocument()

The **ViewMasterDocument** statement switches the active document to master document view. The **ViewMasterDocument()** function returns the following values.

Value	Explanation
0 (zero)	If the active document is not in master document view
-1	If the active document is in master document view

See also

View Statements and Functions

ViewOutline

ViewToggleMasterDocument

ViewMenus()

ViewMenus()

Returns a value that indicates which menu bar is displayed: the full menu bar when a document window is open, or the abbreviated menu bar when no document window is open. You can use **ViewMenus()** as an alternative to **CountWindows()** to determine whether at least one document window is open.

Value	Explanation
0 (zero)	If the full menu bar is displayed
1	If only the File, Help, and application Control menus are displayed; note that other menus may be available if they've been added with **ToolsCustomizeMenuBar**

Note
These values correspond to values allowed for the .MenuType argument when you customize menus with **ToolsCustomizeMenus** and **ToolsCustomizeMenuBar**.

See also

View Statements and Functions

ToolsCustomizeMenus

ToolsCustomizeMenuBar

ViewNormal, ViewNormal()

ViewNormal

ViewNormal()

The **ViewNormal** statement switches the active document to normal view. The **ViewNormal()** function returns the following values.

Value	Explanation
0 (zero)	If the active document is not in normal view
-1	If the active document is in normal view

See also

View Statements and Functions

FilePrintPreview

ViewDraft

ViewMasterDocument

ViewOutline

ViewPage

ViewPage, ViewPage()

ViewPage

ViewPage()

The **ViewPage** statement switches the active document to page layout view. The **ViewPage()** function returns the following values.

Value	Explanation
0 (zero)	If the active document is not in page layout view
-1	If the active document is in page layout view

See also

View Statements and Functions

FilePrintPreview

ViewDraft

ViewMasterDocument

ViewNormal

ViewOutline

ViewRibbon, ViewRibbon()

ViewRibbon [*On*]

ViewRibbon()

The **ViewRibbon** statement displays or hides the Formatting toolbar. This statement is included for compatibility with the previous version of Word.

Argument	Explanation
On	Specifies whether to display or hide the Formatting toolbar: 0 (zero) Hides the Formatting toolbar. 1 Displays the Formatting toolbar. Omitted Toggles the Formatting toolbar on and off.

The **ViewRibbon()** function returns the following values.

Value	Explanation
0 (zero)	If the Formatting toolbar is hidden
-1	If the Formatting toolbar is displayed

See also

View Statements and Functions

ToolsOptionsView

ViewRuler

ViewStatusBar

ViewToolbars

ViewRuler, ViewRuler()

ViewRuler [*On*]

ViewRuler()

The **ViewRuler** statement displays or hides the rulers. If the active document is in outline or master document view, an error occurs.

Argument	Explanation
On	Specifies whether to display or hide the rulers: 0 (zero) Hides the rulers. 1 Displays the rulers. Omitted Toggles the rulers on and off.

The **ViewRuler()** function returns the following values.

Value	Explanation
0 (zero)	If the rulers are hidden
-1	If the rulers are displayed

See also

View Statements and Functions

ToolsOptionsView

ViewRibbon

ViewStatusBar

ViewToolbars

ViewStatusBar, ViewStatusBar()

ViewStatusBar [*On*]

ViewStatusBar()

The **ViewStatusBar** statement displays or hides the status bar.

Argument	Explanation
On	Specifies whether to display or hide the status bar: 0 (zero) Hides the status bar. 1 Displays the status bar. Omitted Toggles the status bar on and off.

The **ViewStatusBar()** function returns the following values.

Value	Explanation
0 (zero)	If the status bar is hidden
-1	If the status bar is displayed

See also

View Statements and Functions

ToolsOptionsView

ViewRibbon

ViewRuler

ViewToolbars

ViewToggleMasterDocument

ViewToggleMasterDocument

Switches the active document from outline view to master document view or from master document view to outline view. If the document is in normal view, page layout view, or print preview, an error occurs.

_____

See also

View Statements and Functions

ViewMasterDocument

ViewOutline

ViewToolbars

ViewToolbars [**.Toolbar =** *text*] [**, .Context =** *number*] [**, .ColorButtons =** *number*] [**, .LargeButtons =** *number*] [**, .ToolTips =** *number*] [**, .Reset**] [**, .Delete**] [**, .Show**] [**, .Hide**]

The arguments for the **ViewToolbars** statement correspond to the options in the Toolbars dialog box (View menu).

Argument	Explanation
.Toolbar	The name of the toolbar you want to reset, delete, display, or hide, as listed in the Toolbars box.
.Context	Specifies a template in which to reset an existing toolbar: 0 (zero) or omitted Normal template 1 Active template
.ColorButtons	If 1, displays color toolbar buttons.
.LargeButtons	If 1, displays enlarged toolbar buttons.
.ToolTips	If 1, displays the button name beneath a button when the mouse pointer is over it.
.Reset	Restores the specified toolbar to its default configuration of buttons.
.Delete	Deletes the specified toolbar.
.Show	Displays the specified toolbar.
.Hide	Hides the specified toolbar.

See also

View Statements and Functions

NewToolbar

ToolsOptionsView

ViewRibbon

ViewRuler

ViewStatusBar

ViewZoom

ViewZoom [**.AutoFit**] [**, .TwoPages**] [**, .FullPage**] [**, .NumColumns =** *number*] [**, .NumRows =** *number*] [**, .ZoomPercent =** *text*]

Changes the magnification for the active document and new documents in the current view. The arguments for **ViewZoom** correspond to the options in the Zoom dialog box (View menu). If the active document is not in page layout view and you specify **.TwoPages**, **.FullPage**, **.NumColumns**, or **.NumRows**, an error occurs.

Argument	Explanation
.AutoFit	Sets magnification so the entire width of the page is visible.
.TwoPages	In page layout view, sets magnification so two entire pages are visible.
.FullPage	In page layout view, sets magnification so the entire page is visible.
.NumColumns	When displaying multiple pages in grid formation, the number of columns in the grid.
.NumRows	When displaying multiple pages in grid formation, the number of rows in the grid.
.ZoomPercent	The percentage of magnification relative to the default display (100 percent).

See also

View Statements and Functions

ViewZoom100

ViewZoom200

ViewZoom75

ViewZoomPageWidth

ViewZoomWholePage

ViewZoom100

ViewZoom100

Switches to normal view and sets magnification to 100 percent for the active document and new documents.

_____

See also

View Statements and Functions

ViewZoom

ViewZoom200

ViewZoom75

ViewZoomPageWidth

ViewZoomWholePage

ViewZoom200

ViewZoom200

Switches to normal view and sets magnification to 200 percent for the active document and new documents.

_____

See also

View Statements and Functions

ViewZoom

ViewZoom100

ViewZoom75

ViewZoomPageWidth

ViewZoomWholePage

ViewZoom75

ViewZoom75

Switches to normal view and sets magnification to 75 percent for the active document and new documents.

_____

See also

View Statements and Functions

ViewZoom

ViewZoom100

ViewZoom200

ViewZoomPageWidth

ViewZoomWholePage

ViewZoomPageWidth

ViewZoomPageWidth

Sets magnification so the entire width of the page is visible.

See also

View Statements and Functions

ViewZoom

ViewZoom100

ViewZoom200

ViewZoom75

ViewZoomWholePage

ViewZoomWholePage

ViewZoomWholePage

Sets magnification so the entire page is visible in page layout view. **ViewZoomWholePage** switches to page layout view if the active document is in another view.

_____

See also

View Statements and Functions

ViewZoom

ViewZoom100

ViewZoom200

ViewZoom75

ViewZoomPageWidth

No Help is available for this WordBasic error message.

WordBasic error

The macro was interrupted

You pressed ESC while the macro was running, or the macro contains a Stop statement. You can prevent ESC from interrupting a macro by using DisableInput. When using **Stop**, you can prevent this message from appearing by using the instruction Stop -1.

WordBasic error 5

Illegal function call

The macro gives an improper or out-of-range value to a function or statement argument. For example, the instruction Shell "Cardfile", 6 generates this error, since the highest value accepted by the second **Shell** argument is 4.

WordBasic error 6

Overflow

The result of a calculation is too large. The following are possible causes for this:

- The result is too large to represent within the range allowed for double-precision floating-point numbers. The largest positive number supported in WordBasic is 1.7976931348623E+308.
- The double-precision floating-point number is too large to convert to an integer.

WordBasic error 7

Out of memory

There is not enough memory to perform the operation. See Freeing up memory.

WordBasic error 9

Subscript out of range

A macro instruction refers to an array element with a subscript outside the dimensions of the array.

WordBasic error 11

Division by zero

The macro contains an expression that attempts to divide a value by 0 (zero).

WordBasic error 14

Out of string space

The macro uses string variables that exceed the 64K of memory allocated for strings passed to external functions (declared with the Declare statement).

WordBasic error 22

Invalid array dimension

A Dim or Redim instruction attempts to dimension an array with an invalid number of elements. For example, the instruction specifies a negative array dimension or attempts to define an array of more than 8143 elements.

WordBasic error 24

Bad parameter

The value for a macro instruction's argument is outside the range of accepted values, as in the following example:

```
ChangeCase 50                '"50" is too large a value
```

WordBasic error 25

Out of memory (stack space)

There is not enough memory to perform the operation. See Freeing up memory.

WordBasic error 26

Dialog needs End Dialog or a push button

The definition for a custom dialog box is missing either an **End Dialog** instruction or a command button. A dialog box definition must include at least one OKButton, CancelButton, or PushButton instruction. For more information, see Begin Dialog...End Dialog.

WordBasic error 28

Directory already exists

A MkDir instruction attempts to create a directory that already exists. To correct this error, use the CountDirectories() and GetDirectory$() functions to determine if a directory name exists before using the MkDir statement to create a subdirectory. For example:

```
Sub MAIN
dir$ = InputBox$("Type directory name to create:")
Dim subdirs$(CountDirectories())
For i = 0 To CountDirectories()
    subdirs$(i) = LCase$(GetDirectory$(i + 1))
    If subdirs$(i) = LCase$(dir$) Then MsgBox "Directory already exists."
Next
End Sub
```

WordBasic error 39

CASE ELSE expected

The expression in a Select Case instruction has no matching case. To correct this error, add a **Case Else** instruction to give Word instructions when it finds no matching case.

WordBasic error 51

Internal error

An internal error occurred in the WordBasic interpreter. Please report this error and the conditions under which it occurred to Word for Windows Technical Support.

WordBasic error 52

Bad file name or number

A file input/output instruction refers to a file number that is out of the acceptable range of 1 through 4. For example, the instruction `Open "C:\TMP\FILES.TXT" For Input As #5` generates this error.

WordBasic error 53

File not found

A statement refers to a file that does not exist or that Word cannot find. For example, if the file C:\TMP\FILES.TXT does not exist, the instruction `Open "C:\TMP\FILES.TXT" For Input As #1` generates this error.

WordBasic error 54

Bad file mode

This message may appear for one of the following reasons:

- The macro contains an Open statement with a file mode other than **Input**, **Output**, or **Append**.
- The macro tries to read from a file opened for output.
- The macro tries to write to a file opened for input.
- The macro refers to a device port that does not exist.

WordBasic error 55

File already open

An Open or Kill instruction specifies an open file, or the macro contains multiple **Open** statements for the same file number or filename. To correct this error, change the file number or filename, or use the Close statement to close the file attached to the file number.

WordBasic error 57

Device I/O error

An error occurred on a device I/O (input/output) operation.

WordBasic error 62

Input past end of file

The macro contains an Input statement that reads from either an empty file or a file in which Word has already read all the data. Use program branching and control instructions to make sure the **Input** statements stay within the file. See Basic File Input/Output.

WordBasic error 64

Bad file name

A macro instruction contains an illegal filename specification. For example, an Open or Kill instruction specifies a filename extension of more than three characters.

WordBasic error 67

Too many files

Open statements in the macro have already opened the limit of four files for input, output, or append. Use the Close statement to close one or more of the files before opening any additional files.

WordBasic error 74

Rename across disks

The macro attempts to rename a file with a new drive designation, an action that Word does not allow.

WordBasic error 75

Path/File access error

The operating system cannot make the correct path-to-filename connection. A network drive may be disconnected.

WordBasic error 76

Path not found

The operating system cannot find the specified path.

WordBasic error 100

Syntax error

The following are some of the causes of syntax errors:

A missing quotation mark

A string must be enclosed by quotation marks. An error is generated if one or both are missing.

A missing, misplaced, or extra comma

Each argument for a statement or function must be separated by a comma. A missing, misplaced, or extra comma generates an error.

A missing period

Each argument for a statement that corresponds to a dialog box must begin with a period. For example, .FileName is an argument for the **FileOpen** statement.

A missing parenthesis at the end of a function

A function name must always be followed by an opening and closing parenthesis. A syntax error is generated if one or both are missing.

A missing reserved word

Some WordBasic instructions include more than one reserved word. For example, an **If** instruction must include the **Then** reserved word.

A reserved word name conflict

WordBasic generates a syntax error if you create a variable name that matches a reserved word. For example, a variable named "Then" conflicts with the reserved word **Then**, which is part of the **If** statement.

WordBasic error 101

Comma missing

The macro interpreter did not find an expected comma in a macro instruction.

WordBasic error 102

Command failed

A macro command failed to complete properly. This failure often occurs because the Cancel button was chosen in an input box or a dialog box displayed with a Dialog instruction. The error can be avoided if a dialog box is displayed with a Dialog() instruction. Use an On Error instruction to prevent an error message from appearing when an input box is canceled.

WordBasic error 103

Dialog record variable expected

An instruction that requires a dialog record variable does not specify a dialog record.

WordBasic error 104

ELSE without IF

The macro contains an **Else** instruction without a corresponding **If** instruction. For more information, see If...Then...Else.

WordBasic error 105

END IF without IF

The macro contains an **End If** instruction without a corresponding **If** instruction. For more information, see If...Then...Else.

WordBasic error 109

INPUT missing

Word expects an Input statement in the macro.

WordBasic error 111

Expression too complex

A macro instruction contains an expression that is too complex. Try breaking up the expression into multiple expressions.

WordBasic error 112

Identifier expected

Word expects an identifier such as a variable, subroutine name, or user-defined function name. For example, the macro includes a Dialog instruction that does not specifiy a dialog record to display.

WordBasic error 113

Duplicate label

This error occurs for one of the following reasons:

- The macro includes two subroutines or user-defined functions with the same name.
- The macro includes two line labels that have the same name (line labels are used with Goto and On Error instructions).

WordBasic error 114

Label not found

This error often occurs because a line label referred to by a Goto or On Error instruction does not begin at the start of a line, does not end with a colon (:), or does not appear in the macro.

WordBasic error 115

Right parenthesis missing

The macro interpreter did not find an expected right (closing) parenthesis in a macro instruction.

WordBasic error 116

Argument-count mismatch

This error is generated when a statement or function has too many or too few arguments. Here are some examples:

```
a$ = Str$()                 'Str$() requires an argument
a$ = Selection$(1)          'Selection$() does not take an argument
MsgBox                      'MsgBox requires an argument
ChDir                       'ChDir requires an argument
```

WordBasic error 117

Missing NEXT or WEND

A **For** or **While** instruction is missing a corresponding **Next** or **Wend** instruction. For more information, see For...Next or While...Wend.

WordBasic error 118

Nested SUB or FUNCTION definitions

A subroutine or function appears inside another subroutine or function. Be sure to include an **End Sub** or **End Function** instruction before beginning a new subroutine or function with the **Sub** or **Function** instruction. For more information, see Sub...End Sub and Function...End Function.

WordBasic error 119

NEXT without FOR

The macro contains a **Next** instruction without a corresponding **For** instruction. For more information, see For...Next.

WordBasic error 120

Array already dimensioned

More than one Dim statement exists for the same array. Use the Redim statement.

WordBasic error 122

Type mismatch

This error occurs when an instruction requires one data type but gets another. Because there are only two data types in WordBasic --- strings and numbers --- the error means that a string was used when a number was required, or vice versa. In the following example, pet is meant to be a string variable, but is missing a dollar sign ($), so Word interprets it as a numeric variable and generates an error:

```
pet = "poodle"          'Should be: pet$ = "poodle"
```

This error is also generated when a function is provided with an argument of the wrong type. Here is an example:

```
a$ = Str$("4")          'Should be: a$ = Str$(4)
```

WordBasic error 123

Undefined dialog record field

This error occurs if the argument for a WordBasic dialog-box--equivalent instruction is misspelled, as in the following example:

```
EditFind "sasquatch", .WhleWord = 1                    'Should be: .WholeWord = 1
```

The error also occurs if an argument that doesn't belong is included, as in the following example:

```
EditFind "skunk", .WholeWord = 1, .Musk = 1                    '"Musk" is not valid
```

WordBasic error 124

Undefined Command, Subroutine, or Function

This error usually occurs when a function or statement name is misspelled. It can also occur if the dollar sign ($) is omitted from a function that returns a string. Each of the following instructions generates this error message:

```
MgsBox "Hello"          'Should be: MsgBox "Hello"
EditFnd "muskrat"       'Should be: EditFind "muskrat"
quote$ = Chr(34)        'Should be: quote$ = Chr$(34)
```

In addition, this error can occur if an instruction that should be two words is typed as one --- for example, `EndDialog` instead of `End Dialog` (though both `EndIf` and `End If` are accepted).

WordBasic error 125

Unexpected end of macro

The macro ended unexpectedly. Make sure each **Sub** instruction has a corresponding **End Sub** instruction and that each **Function** instruction has a corresponding **End Function** instruction.

WordBasic error 126

WEND without WHILE

The macro contains a **Wend** instruction without a corresponding **While** instruction. For more information, see While...Wend.

WordBasic error 127

Wrong number of dimensions

A reference to an array specifies a different number of dimensions than the Dim or Redim instruction that defined the array.

WordBasic error 129

Too many nested control structures

While...Wend, **For...Next**, or **If...Then...Else** statements are nested too deeply for Word to carry out the instructions in the macro. Word supports up to 16 levels of nesting.

WordBasic error 130

SELECT without END SELECT

The macro contains a **Select...Case** instruction without a corresponding **End Select** instruction. For more information, see Select Case.

WordBasic error 131

Illegal REDIM to dialog record

The macro assigns a string or numeric value to a variable and then attempts to use the variable as a dialog record variable. A previously defined string or numeric variable cannot be used as a dialog record variable. Use either a new variable name or an existing dialog variable.

WordBasic error 132

External call caused string overflow

Word allows between 256 bytes and 64K of memory for a string passed to a DLL; the DLL lengthened the string such that it is larger than the memory allowed.

WordBasic error 133

Wrong number or type of arguments for DLL call

The external function declared with a Declare statement requires either a different argument type or a different number of arguments. For example, if the Windows function WinDir is declared with only one argument (it requires two), this error occurs when a macro instruction calls the function.

WordBasic error 134

An argument to a function contained an illegal date or time.

The specified date is outside the acceptable range of December 30, 1899 through December 31, 4095, or the date is specified incorrectly. For example, the month component of a DateSerial() instruction is specified as 13 (the acceptable range is 1 through 12).

WordBasic error 137

The specified path is not a valid path option.

The .Path argument for a **ToolsOptionsFileLocations** instruction is not one of the nine valid path options for WINWORD6.INI. For a list of valid path options, see ToolsOptionsFileLocations.

WordBasic error 138

The current selection cannot be modified by this command.

A SelectCurWord or SelectCurSentence instruction attempts to select the current word or current sentence when more than one word or more than one sentence is selected.

WordBasic error 139

Only one user dialog may be up at any time.

The dialog function for a custom dialog box contains a **Begin Dialog...End Dialog** instruction (or calls a subroutine that contains a **Begin Dialog...End Dialog** instruction). A second custom dialog box cannot be defined or displayed while another custom dialog box is displayed.

WordBasic error 140

Dialog control identifier does not match any current control.

A macro instruction specifies an identifier for a custom dialog box control that is not defined in the dialog box definition. For example, a dialog function includes the instruction DlgFocus "MyPushButton" but the dialog box definition contains no control with the identifier .MyPushButton.

WordBasic error 141

The () statement is not available on this dialog control type.

A macro instruction attempts to perform an impossible action on the specified custom dialog box control. For example, a DlgListBoxArray instruction specifies a text box; **DlgListBoxArray** can change the contents of a list box, drop-down list box, or combo box only.

WordBasic error 142

Specified application is not currently running

An application control statement or function specifies an application that is not running. For example, an AppSendMessage instruction specifies "Windows Help" as the window name, but there is no open application whose name begins with "Windows Help."

WordBasic error 143

The dialog control with the focus may not be disabled or hidden.

A DlgEnable or DlgVisible instruction attempts to disable or hide a control that has the focus. To disable or hide a control, first move the focus to another control, as in the following instructions:

```
If DlgFocus$() = "CheckBox1" Then
DlgFocus "OK"
DlgVisible "CheckBox1", 0
End If
```

WordBasic error 144

Focus may not be set to a hidden or disabled control.

A DlgFocus instruction attempts to move the focus to a hidden or disabled control, an action that Word cannot perform. Use DlgEnable or DlgVisible to enable or display the control before moving the focus to it.

WordBasic error 149

The () command cannot be called as a function.

The specified WordBasic statement does not have a function equivalent. To correct the error, use the statement form of the WordBasic command. For example:

```
x = AppMove(200, 220)
```

should be:

```
AppMove 200, 220
```

WordBasic error 150

Dialog control identifier is already defined.

A custom dialog box definition includes two instructions that specify the same *.Identifier* argument. Identifiers for controls in a custom dialog box must be unique.

WordBasic error 152

This command is not available because no document is open.

A custom dialog box definition includes a FilePreview instruction but no document is open to preview. A document window (or macro-editing window) must be open when a custom dialog box that includes a file preview control is displayed.

WordBasic error 155

The selection does not start in a field.

The macro contains a GetFieldData$() or PutFieldData instruction. When the instruction runs, the selection does not begin in an ADDIN field. **GetFieldData$()** and **PutFieldData** retrieve and store data from ADDIN fields only.

WordBasic error 157

The field cannot contain data.

The macro contains a GetFieldData$() or PutFieldData instruction. When the instruction runs, the selection does not begin in an ADDIN field. **GetFieldData$()** and **PutFieldData** retrieve and store data from ADDIN fields only.

WordBasic error 158

The value of one of the fields is too low.

An argument for an instruction that corresponds to a Word dialog box has a value below the acceptable range. For example, an **InsertBreak** instruction generates this error if the .Type argument is set to --1 (the acceptable range is 0 (zero) to 6).

WordBasic error 159

The value of one of the fields is too high.

An argument for an instruction that corresponds to a Word dialog box has a value above the acceptable range. For example, an **InsertBreak** instruction generates this error if the .Type argument is set to 9 (the acceptable range is 0 (zero) through 6).

WordBasic error 160

Wrong number of parameters

This error is generated when a statement or function has too many or too few arguments (also known as "parameters"). Here are some examples:

```
a$ = Chr$()                 'Chr$() requires an argument
a$ = Str$()                 'Str$() requires an argument
a$ = Selection$(1)          'Selection$() does not take an argument
MsgBox                      'MsgBox requires an argument
ChDir                       'ChDir requires an argument
```

WordBasic error 161

Cannot change dialogs when focus is changing (action 4)

When a custom dialog box is displayed and an *Action* value of 4 is passed to a dialog function, Word cannot display a message box, input box, or dialog box in response. An *Action* value of 4 represents a change of focus in the custom dialog box currently displayed. For more information, see Dialog Function Syntax.

WordBasic error 162

The () command can only be called as a function.

The specified WordBasic command is a function. To correct the error, specify the command as a function. For example:

```
AppInfo$(2)
```

should be:

```
x$ = AppInfo$(2)
```

WordBasic error 163

This statement can only be used when a user dialog is active.

The macro contains an instruction that must be called from within a dialog function while a custom dialog box is displayed. For example, this error occurs if you attempt to use DlgListBoxArray in a dialog box definition instead of a dialog function. The statements and functions that can be used only when a dialog box is displayed all begin with "Dlg." For a complete list, see Dialog Box Definition and Control.

WordBasic error 164

Array variable has not been initialized.

An instruction attempts to assign a value to an array variable that has not been dimensioned in the current subroutine or is not dimensioned as a shared variable. Use the Dim statement to dimension the array.

WordBasic error 500

Cannot initiate link

The macro cannot initiate a DDE link. Possible reasons include the following:

- The other application does not support DDE or has too many active links.
- The other application is not running and Word cannot start it.
- Word does not have enough memory.
- The other application does not currently support the topic specified.
- Word has too many active links.
- The other application is busy.
- The other application is ignoring requests to initiate a DDE link. For example, Microsoft Excel and Microsoft Access both have options to ignore requests to initiate DDE links.

WordBasic error 501

Invalid channel number

The specified DDE channel number is not valid. Make sure the application is still running and that the variable name in the line with the error matches the variable name used in the DDEInitiate instruction.

WordBasic error 502

Application does not respond

The application does not respond to a DDE request. Possible reasons include the following: the other application does not support the DDE request, Word does not have enough memory, or the other application is busy.

WordBasic error 503

Process failed in other application

A DDE statement is incomplete, or the other application is unable to carry out the instruction passed to it by DDEExecute. The DDE statement may also refer to a DDE topic that is not valid---for example, a file that is not open.

WordBasic error 504

Window does not exist

Word cannot find the specified application or document window.

WordBasic error 505

Cannot activate application

Word cannot activate the specified application.

WordBasic error 506

Cannot send keys

Word cannot send the string specified in a SendKeys instruction. The name of a key may be misspelled or a key may be specified that does not exist---for example, {entr} or {F17}.

WordBasic error 508

Other application is busy

The other application is busy and cannot respond to requests from Word.

WordBasic error 509

The () command is not available because ().

The text of WordBasic error 509 varies according to the cause of the error; the message text itself provides the most specific help. Note that many commands are unavailable when a macro-editing window is active and that unavailable menu commands appear dimmed on the menu.

WordBasic error 511

No such macro or command

An instruction such as MacroDesc$() or MacroCopy specifies a macro that does not exist.

WordBasic error 512

Value out of range

The macro specifies a value that Word does not currently recognize. For example, the instruction WindowList 4, which activates the document window numbered 4 on the Window menu, generates this error if only three document windows are open.

WordBasic error 513

String too long

A string argument contains too many characters. A string can hold as many as 65,280 characters, depending on available memory.

WordBasic error 514

Document not open

An instruction refers to a document that is not open. For example, a MacroCopy instruction specifies a destination template that is not open.

WordBasic error 528

Unable to load spelling checker

Word cannot load the spelling checker. To install the spelling checker, exit Word and run Word Setup.

WordBasic error 529

Cannot open dictionary

Word cannot open the dictionary specified in the macro. Make sure the name of the dictionary is correct.

WordBasic error 530

Dialog box description too complex

The macro attempts to display a custom dialog box that contains more controls than Word can currently display. Word does not have a predefined limit on the number of controls a custom dialog box can include. However, if a custom dialog box includes a very large number of controls, there may not be enough memory available to display the dialog box.

WordBasic error 535

Macro cannot be run because it is already running.

A macro called by the active macro attempts to call the active macro. For example, Macro1 calls Macro2, which in turn calls Macro1.

WordBasic error 536

There is no macro with that name.

A macro instruction specifies a macro that does not exist or is not currently available. For example, a ToolsMacro instruction specifies a macro to rename, but there is no such macro in the context specified by the .Show argument.

WordBasic error 537

Unable to run macro specified

A macro instruction attempts to run a macro that is unavailable. For example, this error occurs if a macro uses the ToolsMacro statement to attempt to run itself.

WordBasic error 538

Unable to edit macro specified

A ToolsMacro instruction attempts to edit a macro, but Word cannot perform the action. For example, this error occurs if the specified macro is stored in a template loaded as a global template. (You can run macros stored in loaded global templates, but you cannot edit them unless you open the template in a document window or open a document based on the template.)

WordBasic error 539

Unable to rename macro specified

An instruction unsuccessfully attempts to rename a macro. For example, this error occurs if the macro to rename specified in a ToolsMacro instruction is open or if the new name is already taken by another macro.

WordBasic error 540

Unable to delete macro specified

An instruction unsuccessfully attempts to delete a macro. For example, this error occurs if the macro to delete specified in a ToolsMacro instruction is open or does not exist, or if the template containing the macro is protected. Make sure the .Show argument in the **ToolsMacro** instruction specifies the appropriate context.

WordBasic error 541

Unable to set description of macro specified

A ToolsMacro instruction unsuccessfully attempts to set the description of a macro. For example, this error occurs if the .Show argument is set to 3 (specifying the active template as the context), but the active template is the Normal template. Or the instruction attempts to modify the description of an execute-only macro or a macro in a template that is not open (for example, a loaded global template).

WordBasic error 543

Unable to open specified library

The Declare instruction specifies a dynamic-link library (DLL) that does not exist or cannot be found. A misspelling of the library name is the most common cause of this error. For example, the error occurs if you type "Kernal" when you mean to type "Kernel."

WordBasic error 544

Unable to execute the scroll command; the scrollbar is not active.

The VScroll and HScroll statements require the scroll bars to be displayed. The horizontal and vertical scroll bars are options on the View tab in the Options dialog box (Tools menu). The following example turns the scroll bars on before using the **VScroll** and **HScroll** statements.

```
ToolsOptionsView .HScroll = 1, .VScroll = 1
VScroll 10
HScroll 10
```

WordBasic error 545

The () statement is currently disabled.

The macro attempts to run a statement under circumstances in which it cannot be run. You can use CommandValid() to verify that a statement that corresponds to a Word command or dialog box option is available.

WordBasic error 546

Footnotes or endnotes must start at 1 if numbering is not continuous.

A NoteOptions instruction that specifies continuous numbering for endnotes or footnotes also specifies a starting number other than 1, an action that Word does not allow.

WordBasic error 547

Network Permission Error

The macro attempts to perform an action for which it does not have permission. For example, the macro attempts to delete a file that another user has open on the network.

WordBasic error 549

The specified menu or menu item does not exist.

A macro instruction that returns information about menus specifies a menu or menu item that does not exist. For example, the instruction `a$ = MenuText$(0, 10)` attempts to return the text of the tenth menu, but there are only nine menus.

WordBasic error 551

Word is unable to perform this action because the specified template is locked.

An EditAutoText instruction attempts to delete an AutoText entry stored in a template loaded as a global template. AutoText entries cannot be deleted from global templates (except the Normal template). To delete an AutoText entry in a global template, open the template in a document window or open a document based on the template, and then use the **EditAutoText** statement.

WordBasic error 552

Word is unable to perform this action because the specified template does not exist.

An EditAutoText or SetAutoText instruction attempts to add an AutoText entry to a template other than the Normal template, but no other template is available. For example, this error occurs if the current document is based on the Normal template and the .Context argument is set to 1 (specifying a template other than the Normal template).

WordBasic error 553

Unable to create macro specified

A ToolsMacro instruction unsuccessfully attempts to create or edit a macro. For example, this error occurs if the specified macro already exists as an execute-only macro or the template for which you want to create the macro is protected. This error can also occur if the .Show argument is set to 3 (specifying the active template as the context) but the active template is the Normal template.

WordBasic error 554

No drawing range has been set.

A macro drawing statement or function (such as DrawSelect or DrawCount()) requires a drawing range. Add a DrawSetRange instruction to set a drawing range.

WordBasic error 555

The bookmark specified for the drawing range is invalid.

A DrawSetRange instruction specifies a bookmark that does not exist. Or the bookmark for a previously established drawing range has been deleted, causing a statement that requires a drawing range (such as DrawSelect or DrawCount()) to generate an error.

WordBasic error 556

Wrong drawing object type for this command.

A drawing statement or function designed for a specific type of object is used on an object of a different type. For example, a DrawCountPolyPoints() function is run on a rectangular drawing object. **DrawCountPolyPoints()** is valid for freeform drawing objects only.

WordBasic error 557

Could not insert the drawing object.

The macro attempts to insert a drawing object such as a text box or an arc into an area of the document such as an annotation or footnote pane, an action that Word does not allow.

WordBasic error 558

At least one subdocument in this master document is locked. No changes can be made to any locked subdocuments.

A ToolsRevisions instruction attempts to accept revisions in a master document when at least one subdocument containing unaccepted revisions was locked.

WordBasic error 559

The current selection is a block.

A ToolsReviewRevisions instruction attempts to accept or reject a revision within a block selection, an action that Word does not allow. You can use ToolsRevisions to accept or reject all the revisions in a document, regardless of whether the current selection is a block selection.

Note: Block selections are created by holding down the ALT key and dragging or by using CTRL+SHIFT+F8 and the arrow keys.

WordBasic error 560

The revision marks are not visible.

The macro contains an instruction such as ToolsReviewRevisions .FindNext that depends on revision marks being visible in the document, but revision marks are not displayed. You can use ToolsRevisions to display revision marks.

WordBasic error 561

Document is protected

A macro instruction attempts to change a protected document. For example, this error occurs if the active document is protected for revisions and the instruction ToolsRevisions .AcceptAll runs. You can remove document protection with ToolsUnprotectDocument.

WordBasic error 562

ToolsGrammarStatisticsArray cannot be run on a document that contains more than one language format.

A ToolsGrammarStatisticsArray instruction was run on a document that contains at least two different language formats. You can use ToolsLanguage to set and remove language formats.

WordBasic error 563

The document is not a master document.

The macro contains a statement that is valid only when the active document is a master document (for example, LockDocument or GoToNextSubdocument). You can use the ViewMasterDocument and CreateSubdocument statements to create a master document.

WordBasic error 564

There are no subdocuments in that direction.

A macro contains a GoToPreviousSubdocument instruction but the selection is already within or above the first subdocument in the master document. Or the macro contains a GoToNextSubdocument instruction but the selection is already within or below the last subdocument.

WordBasic error 565

The specified document is not in the Add-in list.

An instruction specifies a template or add-in that does not appear in the list of global templates and add-ins in the Templates And Add-ins dialog box (File menu). For example, a DeleteAddIn instruction specifies a template that does not appear in the list.

WordBasic error 566

The specified Word library cannot be unloaded because it is in use.

An instruction attempts to unload a Word add-in library (WLL), but the WLL is in use and cannot be unloaded.

WordBasic error 567

Cannot add more than 25 items in the dropdown list box

A drop-down form field can hold no more than 25 items. If the form field already includes 25 items, an AddDropDownItem instruction produces this error.

WordBasic error 568

The specified font doesn't exist

The font specified in the .CommandName argument of a ToolsCustomizeMenus, ToolsCustomizeKeyboard, or AddButton instruction does not exist or is not currently available.

WordBasic error 569

PatternMatch and SoundsLike parameters cannot both be set to 1

Just as Word does not allow selecting both the Use Pattern Matching and Sounds Like check boxes in the Find and Replace dialog boxes (Edit menu), Word prevents setting both .PatternMatch and .SoundsLike to 1 in EditFind and EditReplace WordBasic instructions.

WordBasic error 570

Cannot sort arrays with more than two dimensions

A SortArray instruction attempts to sort an array with more than two dimensions. **SortArray** sorts only one-dimensional or two-dimensional arrays.

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

Activate Example

This example stores the name of the active window, opens a document, and then reactivates the original window:

```
firstwin$ = WindowName$()
FileOpen "CHAP2.DOC"
Activate firstwin$
```

Activate

¬Example

Activate *WindowTitle$* [**,** *PaneNum*]

Activates the window with the specified *WindowTitle$*. You can use the **WindowName$()** function to store the name of a window in a variable and then use this variable with **Activate** when you need to activate that window.

WindowTitle$ must match a window name on the Window menu. Note that window titles change when you change directories, so use **Activate** with caution. Window titles for documents stored in the current directory contain filenames only. Window titles for documents not stored in the current directory contain the full path in addition to the filename.

Argument	Explanation
WindowTitle$	The name of the window to activate, as it appears on the Window menu
PaneNum	The number of the pane to activate: 1 or 2 Top pane 3 or 4 Bottom pane

See also

Windows Statements and Functions

DocSplit

NextWindow

OtherPane

PrevWindow

WindowList

WindowName$()

Window*Number*

WindowPane()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

ClosePane Example

This example closes the lower pane in the active document if the window is split:

```
If DocSplit() Then ClosePane
```

ClosePane

¬Example

ClosePane

Closes a pane. Use this statement to close the lower pane in a split document window, a footnote pane, or any other kind of pane. **ClosePane** generates a WordBasic error if no pane is open in the active document. Note that **ClosePane** does not close a document window.

See also

Windows Statements and Functions

DocSplit

OtherPane

WindowPane()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

CountWindows() Example

This macro arranges windows side by side instead of stacking them as the Arrange All command (Window menu) does:

```
Sub MAIN
yours$ = WindowName$()                    'Note which window is active
If DocMaximize() Then DocRestore          'Can't resize if maximized
fullwidth = Val(AppInfo$(6))              'Get workspace width
fulldepth = Val(AppInfo$(7))              'Get workspace depth
width = fullwidth / CountWindows()
For wnd = 1 To CountWindows()             'Move each window, resize it
    x = width *(wnd - 1)
    If DocMinimize() Then DocRestore      'Can't resize if minimized
    DocMove x, 0
    DocSize width, fulldepth - 1
    NextWindow
Next wnd
Activate yours$                           'Reactivate original window
End Sub
```

CountWindows()

¬Example

CountWindows()

Returns the number of open document and macro-editing windows, which is also the number of windows listed on the Window menu.

_____

See also

Windows Statements and Functions

WindowName$()

Window*Number*

DocMaximize, DocMaximize()

DocMaximize [*State*]

DocMaximize()

The **DocMaximize** statement maximizes or restores all document windows according to the value of *State*: 1 maximizes windows and 0 (zero) restores windows. If *State* is not specified, **DocMaximize** toggles document windows between restored and maximized states.

The **DocMaximize()** function returns the following values.

Value	Explanation
-1	If the document windows are maximized
0 (zero)	If the document windows are not maximized

See also

Windows Statements and Functions

AppMaximize

DocMinimize

DocRestore

DocMinimize, DocMinimize()

DocMinimize

DocMinimize()

The **DocMinimize** statement shrinks the active document window to an icon.

The **DocMinimize()** function returns the following values.

Value	Explanation
-1	If the active document window is minimized
0 (zero)	If the active document window is not minimized

See also

Windows Statements and Functions

AppMinimize

DocMaximize

DocRestore

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocMove Example

This example moves the document window to the position 20 points to the right and 40 points down from the upper-left corner of the workspace:

```
If DocMaximize() = 0 Then DocMove 20, 40
```

DocMove

¬Example

DocMove *HorizPos*, *VertPos*

Moves the active document window or icon to the specified location. If the active document is maximized, Word cannot perform this action and generates an error.

Argument	Explanation
HorizPos, *VertPos*	The horizontal (*HorizPos*) and vertical (*VertPos*) distance from the upper-left corner of the workspace to the upper-left corner of the document window, in points (72 points = 1 inch). Negative values are allowed.

See also

Windows Statements and Functions

AppMove

DocSize

DocWindowPosLeft

DocWindowPosTop

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocRestore Example

This example restores document windows if they are maximized:

```
If DocMaximize() <> 0 Then DocRestore
```

DocRestore

¬Example

DocRestore

Restores all document windows from a maximized state or restores a minimized document window. Unlike **DocMaximize**, which maximizes the document windows if restored and restores them if maximized, **DocRestore** only restores document windows. It generates an error if the active document is already restored.

There is no corresponding function for **DocRestore**. As the example for this entry demonstrates, however, you can use the **DocMaximize()** function to determine whether document windows are restored.

_____

See also

Windows Statements and Functions

AppRestore

DocMaximize

DocMinimize

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocSize Example

This example ensures that document windows are restored and then sizes the active document window to half the width of the workspace:

```
If DocMaximize() or DocMinimize() Then DocRestore
DocSize(Val(AppInfo$(6)) / 2), Val(AppInfo$(7))
```

DocSize

¬Example

DocSize *Width*, *Height*

Sizes the active document window to the specified *Width* and *Height*, in points (72 points = 1 inch). If the active document is maximized or minimized, Word cannot perform this action and generates an error. Note that this statement must have arguments for both the width and height of the window. To size only the width or height of a window, use **DocWindowWidth** or **DocWindowHeight**.

_____

See also

Windows Statements and Functions

AppInfo$()

AppSize

DocWindowHeight

DocWindowWidth

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocSplit Example

This example splits the active document window in the middle:

```
DocSplit 50
```

DocSplit, DocSplit()

¬Example

DocSplit *Percentage*

DocSplit()

The **DocSplit** statement splits the active document window at the given height, expressed as a percentage of the distance between the top and bottom of the document area.

The **DocSplit()** function returns the split position as a percentage of the active document area's height or returns 0 (zero) if the document window is not split.

The **DocSplit** statement accepts values between 0 (zero) and 100, but values near the ends of the range may not split the document window, depending on the screen elements that are displayed and the size of the window. Using the **WindowPane()** function, you can determine whether a window is split.

_____

See also

Windows Statements and Functions

ClosePane

OtherPane

WindowPane()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocWindowHeight Example

This example restores the active document window if it is maximized or minimized, moves it to the upper-left corner of the workspace, and then sizes the window to 267 points high:

```
If DocMaximize() or DocMinimize() Then DocRestore
DocMove 0, 0
DocWindowHeight 267
```

DocWindowHeight, DocWindowHeight()

¬Example

DocWindowHeight *Height*

DocWindowHeight()

The **DocWindowHeight** statement sets the active document window to the specified *Height*, in points (72 points = 1 inch), without affecting its width. If the active document is maximized or minimized, Word cannot perform this action and generates an error. The **DocWindowHeight()** function returns the height of the active document window in points.

See also

Windows Statements and Functions

AppWindowHeight

DocSize

DocWindowWidth

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocWindowPosLeft Example

This example displays the horizontal and vertical positions of the active document window relative to the border of the workspace:

```
horizpos = DocWindowPosLeft()
vertpos = DocWindowPosTop()
MsgBox "Points from left:" + Str$(horizpos) + Chr$(13) + \
    "Points from top:" + Str$(vertpos), "Doc Window Position"
```

DocWindowPosLeft, DocWindowPosLeft()

¬Example

DocWindowPosLeft *Position*

DocWindowPosLeft()

The **DocWindowPosLeft** statement positions the active document window or icon so that it is *Position* points from the left border of the workspace. If the active document is maximized, Word cannot perform this action and generates an error. The **DocWindowPosLeft()** function returns the horizontal position in points.

See also

Windows Statements and Functions

AppWindowPosLeft

DocMove

DocSize

DocWindowHeight

DocWindowPosTop

DocWindowWidth

DocWindowPosTop, DocWindowPosTop()

DocWindowPosTop *Position*

DocWindowPosTop()

The **DocWindowPosTop** statement positions the active document window or icon so that it is *Position* points from the upper border of the workspace. If the active document is maximized, Word cannot perform this action and generates an error. The **DocWindowPosTop()** function returns the vertical position in points. For an example, see DocWindowPosLeft Example.

See also

Windows Statements and Functions

AppWindowPosTop

DocMove

DocSize

DocWindowHeight

DocWindowPosTop

DocWindowWidth

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

DocWindowWidth Example

This example maximizes the Word window and restores document windows. If there are two open document windows, the windows are each sized to half the screen width and are placed side by side.

```
If AppMaximize() = 0 Then AppMaximize
If DocMaximize() Or DocMinimize() Then DocRestore
If CountWindows() = 2 Then
    DocMove 0, 0
    DocWindowWidth 240
    NextWindow
    If DocMinimize() Then DocRestore
    DocMove 240, 0
    DocWindowWidth 240
End If
```

DocWindowWidth, DocWindowWidth()

¬Example

DocWindowWidth *Width*

DocWindowWidth()

The **DocWindowWidth** statement sets the active document window to the specified *Width*, in points (72 points = 1 inch), without affecting its height. If the active document is maximized or minimized, Word cannot perform this action and generates an error. The **DocWindowWidth()** function returns the width of the active document window in points.

See also

Windows Statements and Functions

AppWindowWidth

DocSize

DocWindowHeight

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

OtherPane Example

This example inserts a footnote and then returns the insertion point to the footnote reference mark. In page layout view, where footnotes are not inserted in a separate pane, the instruction GoBack returns the insertion point to the reference mark. In normal view or outline view, the instruction OtherPane returns the insertion point.

```
FntText$ = "This is the footnote text"
InsertFootnote
Insert FntText$
If ViewPage() Then
    GoBack
Else
    OtherPane
End If
```

OtherPane

¬Example

OtherPane

Moves the insertion point to its most recent position in the next pane of the active window. **OtherPane** is often used in macros that open the annotation or footnote panes.

_____

See also

Windows Statements and Functions

ClosePane

DocSplit

WindowPane()

Window()

Window()

Returns a number that corresponds to the position of the active window on the Window menu, where 1 corresponds to the first position, 2 to the second position, and so on. If there are no open windows, **Window()** returns 0 (zero).

Word lists windows on the Window menu in alphabetic order. Note that changing the current directory can affect window titles and therefore their order on the Window menu. Window titles for documents stored in the current directory contain the filename only, while window titles for documents not stored in the current directory contain the full path as well as the filename.

_____

See also

Windows Statements and Functions

WindowList

WindowName$()

Window*Number*

WindowPane()

WindowArrangeAll

WindowArrangeAll

Arranges all open windows so they do not overlap.

_____

See also

Windows Statements and Functions

DocMove

DocRestore

DocSize

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

WindowList Example

This example activates the first window containing the document TEST.DOC:

```
numwin = CountWindows()
If numwin <> 0 Then
    i = 1
    While i <= numwin And leave <> 1
        winname$ = WindowName$(i)
        If InStr(winname$, "TEST.DOC") Then leave = 1
        If leave <> 1 Then i = i + 1
    Wend
End If
If InStr(winname$, "TEST.DOC") Then
    WindowList i
Else
    MsgBox "There is no window containing TEST.DOC."
End If
```

WindowList

¬Example

WindowList *Number*

Activates a window listed on the Window menu. The instruction WindowList 1 activates the first window in the list, WindowList 2 activates the second window, and so on through the number of open windows. If no windows are listed, or if *Number* is greater than the number of open windows, an error occurs.

See also

Windows Statements and Functions

CountWindows()

Window()

WindowName$()

Window*Number*

WindowPane()

WindowName$()

WindowName$(*Number***)**

Returns the title of the open window listed at position *Number* on the Window menu, where 1 corresponds to the first position, 2 to the second position, and so on. If *Number* is 0 (zero) or omitted, **WindowName$()** returns the title of the active window. Note that changing the current directory can affect window titles and therefore their order on the Window menu. Window titles for documents stored in the current directory contain filenames only, while window titles for documents not stored in the current directory contain the full path as well as the filename.

For an example, see WindowList Example.

See also

Windows Statements and Functions

CountWindows()

Window()

WindowList

Window*Number*

WindowPane()

WindowNewWindow

WindowNewWindow

Opens a new window containing the active document. Word adds a colon (:) and a number to the titles of windows containing a document that is open in more than one window. For example, if the window title of the active document is TEST.DOC and you run **WindowNewWindow**, Word opens a new window titled TEST.DOC:2 and changes the original window title to TEST.DOC:1.

See also

Windows Statements and Functions

DocSplit

WindowArrangeAll

WindowName$()

Window*Number*

Window*Number*

Activates a window listed on the Window menu. **Window1** activates the first window in the list, **Window2** activates the second window, and so on through **Window9**. If no windows are listed, or if *Number* is greater than the number of windows listed, an error occurs.

Note

You cannot use a variable in place of *Number*; you must use an integer. **WindowList** provides the same functionality as **Window***Number* but accepts a numeric variable to specify the window to activate.

See also

Windows Statements and Functions

Activate

CountWindows()

Window()

WindowList

WindowName$()

WindowPane()

{ewc shareres, T3EWClass, $$button:WordBASICClose}{ewc shareres, T3EWClass, $$button:WordBASICCopy}{ewc shareres, T3EWClass, $$button:WordBASICPrint}{ewc shareres, T3EWCLASS, $$+button:WordBASICGreyBar}

WindowPane() Example

This example moves the insertion point to the top pane of the active document if the active window is split and the insertion point is in the bottom pane:

```
If WindowPane() = 3 Then OtherPane
```

WindowPane()

¬Example

WindowPane()

Returns the following values.

Value	Explanation
1	If the active window is not split or if the insertion point is in the top pane of the active window
3	If the insertion point is in the bottom pane of the active window (for example, the footnote pane, the annotation pane, or the lower of two document panes)

See also

Windows Statements and Functions

DocSplit

OtherPane

ViewAnnotations()

ViewFootnoteArea()