Contents

For Help on Help, Press F1

Introduction to Macros

If you are new to macros and want to learn how to create your own, start with the section, Macro Fundamentals. You'll learn how to record actions with WSWin and save them as a macro. You'll also learn how to assign macros to keystrokes, menus, and Toolbox buttons.

If you are ready to start programming macros, skip to Macro Programming. This section describes how to plan and program your own macros from scratch using good programming techniques. The sections, Advanced Macro Techniques and Debugging Macros, teach you to create dialog boxes that appear when a macro runs, use internal Windows functions in your macros, and fix errors.

Once you've learned the basics of macro programming, refer to the the last section, Macro Language Reference, for the syntax of all the subroutines and functions in the WSWin macro system.

Related Topics

What Are Macros?

Recordings vs. Macros

Macro Limitations

What Are Macros?

WSWin macros increase the power and flexibility of WSWin by combining many actions in a single command. Macros automate repetitive tasks or simplify complicated actions. Macros can prompt for input, display messages, and interact with other Windows programs.

Macros are external files that contain a series of commands to perform actions inside WSWin. Macros are executed line by line, with each line giving instructions to WSWin on what actions to perform.

The WSWin macro system is based on the BASIC programming language. If you're already familiar with BASIC, you'll find WSWin macros easy to read and understand. The Macro Language Reference section describes the individual subroutines that make up the WSWin macro language.

Recordings vs. Macros

WSWin can record your actions so that you can play them back later. These recorded actions are called "recordings." A recording exists in the computer's memory and is not saved to a file until you request it. When you exit WSWin, the recording will be lost unless you save it in a macro file. For more information on recordings, see Recording and Playing Back Actions.

A macro is a separate file of commands. It exists on your hard disk in a special directory with other macros. It is loaded in memory only when you run the macro. As soon as the macro stops running, it is unloaded from memory, but remains on your hard disk until it is needed again.

Use recordings when you need a quick way to remember a series of actions so that you can play them back. You can also use recordings as a launching pad for writing your own macros. After you have recorded the steps, you can edit the recording, save it in a macro file, and customize it.

Macro Limitations

Macros are a powerful feature of WSWin, but they have limitations:

- You cannot use a macro to call another macro.
- You cannot modify the contents of graphics, EPS (encapsulated PostScript), or OLE (object linking & embedding) frames with macros.

Macro Fundamentals

Recording and Playing Back Actions

The simplest way to automate a sequence of actions is to record them. WSWin keeps track of most of the actions and stores them in a special area of memory called a "recording." The recording exists as long as WSWin is running. If you want to be able to use the recording each time you run WSWin, then you must save it as a macro file.

To turn recording on, choose Macro ▸Start Recording or click the Record button on the Toolbox (the cassette tape with the red record indicator below it). The letters "REC" appear on the Status Bar when recording is on.
Perform the actions you want to record. If you must move the insertion point while you are recording, you should use keystrokes rather than the mouse. Some mouse actions are recorded but they may not give the desired results in other documents.
To turn recording off, choose Macro ▸Stop Recording or click the Record button on the Toolbox. The recording is stored in memory and can be played back.

To play back the recording, choose Macro ▸Playback Recording or click the Playback button on the Toolbox (the cassette tape with the blue playback indicator below it). The actions in the recording are repeated in the active document.
Some recordings depend on the position of the insertion point. Remember to position the insertion point in the correct location before playing back those recordings.
To save and edit your recordings, choose either Macro ▸Edit Recording or Macro ▸Save Recording. These commands are described in Creating and Editing Macros..

Running Macros

You "play back" a recording and "run" a macro. In many cases, the results are identical. The difference is that a macro is a separate file (much like an external application program) that must be executed.

To run a macro, choose Macro▸Run Macro. Select a macro to run from the dialog box. You can also use the Macro Editor to run a macro.
If you run a macro frequently, you can assign the macro to a keystroke, menu, or Toolbox button.

Related Topics

Assigning a Macro to a Keystroke
Assigning a Macro to a Menu
Assigning a Macro to a Toolbox Button

Managing Macro Files

WSWin macros are stored in a subdirectory of the WSWin application directory. You can specify the default macro directory in the Preferences dialog box. You usually keep all your macros in this directory. The Macro Editor also uses this directory when you open or save macro files.

WSWin adds the extension .WMC when you save a macro. You should also use this extension when you create your own macros.

A macro file has two parts. The first part is the text that makes up the macro instructions. It is called the macro source and appears in the Macro Editor. The second part is the compiled, or object code, portion that is created automatically by the macro compiler. The compiled object code executes when you run a macro. The macro compiler updates the compiled object code when you change the macro source.

Related Topics

Sharing Macro Files with Other Users

Sharing Macro Files with Other Users

You can share macros with other WSWin users by copying macro files to floppy disks or shared network directories.

When you create a macro, include comment lines at the beginning of the macro with a description of the macro and your name. For more information, see Creating and Editing Macros.

Creating and Editing Macros

Creating a Macro from a Recording

If you have a recording that you want to use when you start WSWin, you must save it as a macro. To do this, choose Macro▸Save Recording. In the Save dialog box, assign a name to the macro file. Use a name that you can remember. To run the macro, choose the Macro
▸Run Macro command, or assign the macro to a keystroke, menu, or Toolbox button.
To modify the recording before you save it, choose the Macro▸Edit Recording command. The Macro Editor displays the current recording as macro instructions. You can add a description to the macro that displays in the Status Bar when you select a Toolbox button or menu item assigned to the macro. The description also appears in the dialog boxes for customizing the keystrokes, menus, and Toolbox. To assign a filename to the macro, choose the File
▸Save command in the Macro Editor menu.

Macro Editor Basics

The Macro Editor is a separate application that is included with WSWin designed for editing macros.

In the Macro Editor, text does not wrap automatically to the next line. You can type lines up to 255 characters, and the window scrolls to the right as needed.

Use the arrow keys to move around in the macro window. Press the Ctrl key with the left and right arrow keys to move from word to word. Press the Shift key to extend the selection.

You can edit more than one macro file at a time. Each macro appears in a separate window. Use the commands on the Window menu in the Macro Editor to arrange the macro windows or to switch to a different window. You can also use Ctrl+Tab and Ctrl+Shift+Tab to switch to the next or previous macro window.

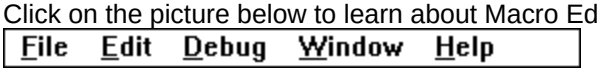
An indicator at the bottom of the macro window shows the current line number. This can help you find and fix errors.

Related Topics

Macro Editor Menus

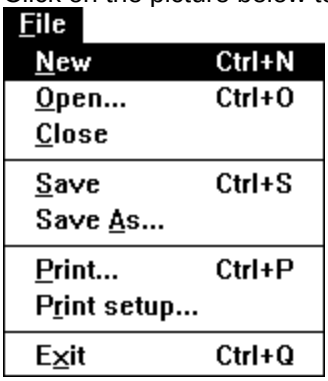
Macro Editor Menus

Click on the picture below to learn about Macro Editor commands.



File Menu

Click on the picture below to learn about File Menu commands.



New Command

Choose New from the File menu to open an untitled macro window.

Open Command

Choose Open from the File menu to open a macro file.

Close Command

Choose Close from the File menu to close a macro window. If your changes have not been saved, a confirmation message appears.

Save Command

Choose Save from the File menu to save a macro. If the macro window is untitled, the Save As dialog box appears.

Save As Command

Choose Save As from the File menu to save a macro with a new name or to save an untitled macro.

Print Command

Choose Print from the File menu to print the current macro. For more information, see Printing Macros.

TIP If the lines in your macro are long, use the Print Setup command to set your printer to landscape orientation before printing a macro.

Print Setup Command

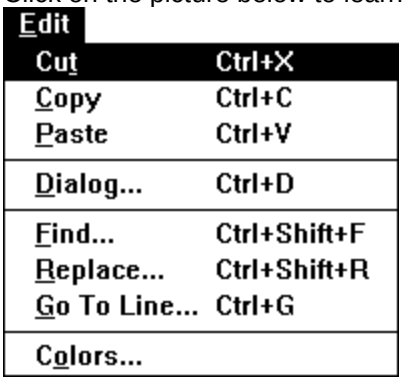
Choose Print Setup from the File menu to change printer options.

Exit Command

Choose Exit from the File menu to exit the Macro Editor.

Edit Menu

Click on the picture below to learn about Edit Menu commands.



Cut Command

Choose Cut from the Edit menu to delete selected text and place it on the Clipboard.

Copy Command

Choose Copy from the Edit menu to place selected text on the Clipboard.

Paste Command

Choose Paste from the Edit menu to insert text on the Clipboard at the insertion point.

Dialog Command

Choose Dialog from the Edit menu to create a dialog box for a macro. To edit an existing dialog box, select the dialog box statements in the macro and choose the command.

Related Topics

Dialog Editor Tools

Dialog Editor Tips

Find Command

Choose Find from the Edit menu to locate text in the macro.

Replace Command

Choose Replace from the Edit menu to find a text string in the macro and replace it with another string.

Go To Line Command

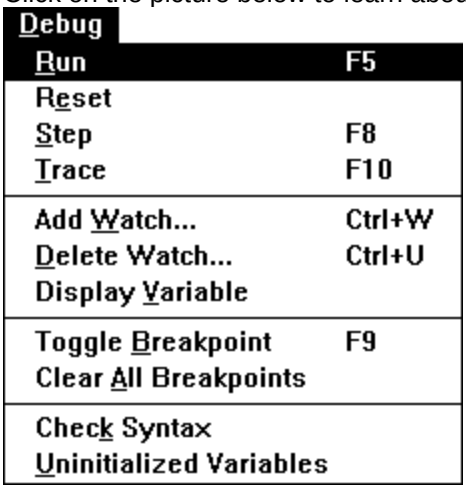
Choose Go To Line from the Edit menu to go to a specific line in the macro.

Colors Command

Choose Colors from the Edit menu to change the default colors for the Macro Editor.

Debug Menu

Click on the picture below to learn about Debug Menu commands.



Related Topics

Debugging Macros

Run Command

Choose Run from the Debug menu to run the current macro.

Reset Command

Choose Reset from the Debug menu to stop the current macro and reset all variables to the default values. You must pause the macro by stepping through a macro or setting a breakpoint before you can reset the macro.

Step Command

Choose Step from the Debug menu to execute the current line in the macro. The Macro Editor highlights the next line in the macro.

Use Step when you want to step through your custom functions and subroutines. To stay in the main body of the macro, use the Trace command.

Trace Command

Choose Trace from the Debug menu to execute the current line in the macro. The Macro Editor highlights the next line in the macro.

Use Trace when you want to stay in the main body of the macro. To step through your custom functions and subroutines, use the Step command.

Add Watch Command

Choose Add Watch from the Debug menu to monitor the value of a variable in the macro. As you step through a macro, the value in the Watch window changes as the variable changes.

Delete Watch Command

Choose Delete Watch from the Debug menu to remove a watch from the Watch window.

Display Variable Command

Choose Display Variable to display the current value of a variable in a dialog box.

Toggle Breakpoint Command

Choose Toggle Breakpoint to set a breakpoint on a line in the macro. Choose the command again to clear the breakpoint.

Clear All Breakpoints Command

Choose Clear All Breakpoints from the Debug menu to clear all breakpoints in the macro.

Check Syntax Command

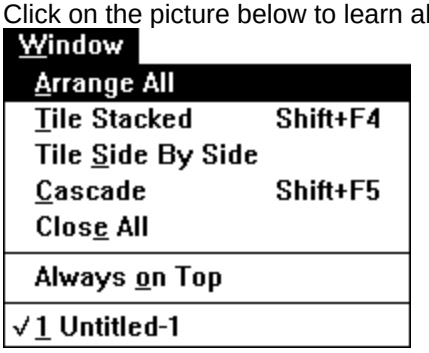
Choose Check Syntax to check for errors in the macro. If errors are found, error messages appear in a window. Double-click on an error message to go to the line containing the error.

Uninitialized Variables Command

Choose Uninitialized Variables from the Debug menu to check for variables that have not been initialized. When you choose Check Syntax, the Macro Editor displays error messages when a variable has not been initialized to a value before it is used.

Window Menu

Click on the picture below to learn about Window Menu commands.



Arrange All Command

Choose Arrange All from the Window menu to arrange all open macro windows. Minimized windows are arranged at the bottom of the Macro Editor window.

Tile Stacked Command

Choose Tile Stacked from the Window menu to tile open windows one on top of another.

Tile Side By Side Command

Choose Tile Side By Side from the Window menu to tile open windows next to each other.

Cascade Command

Choose Cascade from the Window menu to overlap open windows.

Close All Command

Choose Close All from the Window menu to close all open windows. A message appears if changes to a macro have not been saved.

Always on Top Command

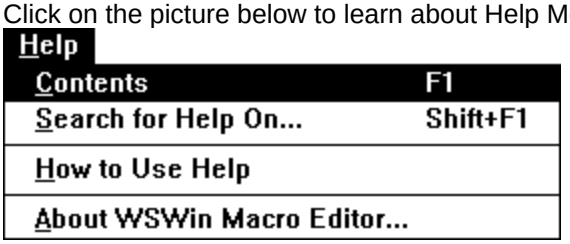
Choose Always on Top from the Window to keep the Macro Editor window visible even when another application is active. This is useful when you are debugging a macro. Choose the command again to turn off the setting.

Open Windows List

The list of open macro windows appears on the bottom of the Window menu. Choose one of the numbered windows to activate that window.

Help Menu

Click on the picture below to learn about Help Menu commands.



Contents

Choose Contents from the Help menu to display the Contents page of the on-line help.

Search for Help On

Choose Search for Help On from the Help menu to look for help on a specific word or phrase.

TIP If the insertion point is on a macro keyword, press Shift+F1 to display the help topic for that keyword.

How to Use Help

Choose How to Use Help from the Help menu to display the Windows Help file.

About WSWin Macro Editor

Choose About WSWin Macro Editor to display information about the Macro Editor.

Editing Techniques

In addition to the standard Windows commands, Cut and Copy, that work on selected text, you can cut or copy the current line in the Macro Editor without selecting it first. Because macros are always executed line by line, you can rearrange the order of the macro instructions.

To cut the current line to the Clipboard, press the minus key (-) on the numeric keypad. To copy the current line, press the plus key (+) on the numeric keypad. If you select text, these keys cut or copy the selected text.

To paste the line, move the insertion point to the new location and choose Paste from the Edit menu. The line from the Clipboard is pasted above the line containing the insertion point.

To find text in the macro window, use the Edit▸Find command.
To replace one or more occurrences of one text string with another, use the Edit▸Replace command.
To go to a specific line in the macro, use the Edit▸Go To Line command.

Using On-Line Help

You can copy a programming example in the help text by clicking the Copy button in the Help window. The Copy dialog box appears. Select the example text in the edit box and click Copy. Position the insertion point in the macro and choose Paste from the Edit menu. You can then modify the example to fit your needs.

To go to the Macro Language Reference, click the Reference button in the Help window.

Printing Macros

To print a macro, choose File▸Print. The Macro Editor will print the macro in the Courier New 10-point font.
If your macro has lines longer than 80 characters, use the File▸Print Setup command to change page orientation to landscape. The Macro Editor does not automatically wrap long lines when printing.

Macro Programming

Introduction to Programming

If you are already familiar with the BASIC programming language, you can read this section quickly, and then go to Advanced Techniques to learn about the powerful capabilities of the WSWin macro system.

If you are new to BASIC programming, read this section carefully. You'll learn to write your own macros and modify your recordings.

Use macros when you want to:

- get user input during the macro
- control the flow of the macro based on specific criteria
- handle a variety situations without user intervention

Related Topics

StarBASIC vs. WSWin Subroutines and Functions

StarBASIC vs. WSWin Subroutines and Functions

The WSWin macro language is made up of two sets of macro instructions: StarBASIC and WSWin instructions. StarBASIC instructions, which are not part of the WSWin command set, are used to display dialog boxes, obtain a keystroke, or modify a string. A macro containing only StarBASIC instructions can run even if WSWin is not running.

WSWin instructions are part of the WSWin command structure. When a macro contains one of the WSWin commands, the macro system communicates with WSWin to get or change information about the current document, for example, to modify a paragraph style. A macro that contains WSWin instructions, as most macros do, can run only when WSWin is running.

In Help, StarBASIC instructions are in uppercase, such as LEFT$, RGB, and MESSAGEBOX. WSWin instructions are in initial caps, such as FileOpen, EditCut, and InsertFrame.

Macro Language Conventions

You can use the following conventions when you write your own macros:

- Using the Two Types of Variables
- Using Arrays
- Macro Parameters
- Functions vs. Subroutines
- Macro System Is Not Case Sensitive
- Macro Comments
- Formatting a Macro for Readability
- Improving Macro Running Speed

Two Types of Variables

To store information in a macro, you use a variable. The contents of a variable change when you assign a value to the variable, or when you pass the variable to an instruction that can modify the value.

The WSWin macro system has two types of variables - variables that hold numbers and variables that hold strings. Numeric variables are always whole numbers. They can range from -2,147,482,648 to 2,147,482,647.

To perform mathematic operations on decimal numbers, multiply them by a multiple of 10 (such as 1000, if you know that all the numbers will have no more than three decimal places), perform the operation, and then divide by the same multiple of 10 to obtain the result. A numeric variable always has a percent sign (%) as the last character of the variable name, such as **pointSize%**.

String variables can hold up to 32,765 characters of text and can contain substrings that specify WSWin formatting, such as a font, paragraph style, or bold. These substrings are valid only for specific WSWin functions. String variables always have a dollar sign ($) as the last character of the variable name, such as **fontName$**. String values that are assigned to string variables must be enclosed in double quotes, as shown in the example below.

```
stringVar$ = "This is a sample string."
```

Related Topics

WSWin Text

Arrays

Arrays are a special type of variable containing more than one of the same variable type. For example, if you want to get information on the paragraph styles in a document, you could use an array of 255 strings to hold all the style names (a document can have a maximum of 255 paragraph styles).

Arrays also use percent and dollar signs to indicate the type of array. An array of one hundred strings could be called **stringArray$(100)**. An array of fifty numbers could be called **numberArray%(50)**.

To use an array, you must address a specific element of the array. For example, to use the fifth string in the array mentioned above, you would use **stringArray$(5)**. You can use numeric variables to address a sequence of array elements, as in the following example that assigns the numbers 1 through 50 to the corresponding elements of a numeric array:

```
FOR i% = 1 TO 50
   numberArray%(i%) = i%
NEXT i%
```

Arrays are created with the DIM (for "dimension") statement.

Macro Parameters

Macro parameters are variables, constants, and literal values that provide additional information to functions and subroutines. For example, to open a file, you must specify a filename.

WSWin uses a default value for parameters that are not included in the parameter list (optional parameters). When you omit parameters, you separate each omitted parameters with commas. For example:

```
DefineStyle "BodyText",
DefineStyleFont "Arial", 120,, ,, ,, ,, ,
EndDefineStyle
```

The commas in the DefineStyleFont statement "stand in" for the omitted parameters.

Related Topics

Macro Parameter Values

WSWin Text

Macro Parameter Values

Macro parameters are either strings or integers. The Macro Language Reference indicates which parameter type with the percent sign and dollar sign symbols.

If a macro parameter is either TRUE or FALSE, use 1 to represent TRUE and 0 to represent FALSE. ON (1) and OFF (0) have the same values.

Parameters that represent measurement, such as point size, frame size and position, use decipoints to specify the measurement. A decipoint is one-tenth of a point. There are 72 points to an inch, so there are 720 decipoints to an inch. If you want to use measurements other than decipoints in your macros, you can use the conversion function TO_DP.

WSWin Text

Some macro statements allow you to specify special string parameters. For example, to find bold text, include a "bold on" tag at the beginning of the search string. In a macro statement, use the special string "<Bd 1>" to turn bold on. This type of string is called a WSWin text string. Some of the macro statements that use WSWin text strings are: EditFind, EditReplace, InsertIndexEntry, and TypeText.

These special strings can be combined using semicolons to separate each string. For example, to change the font to Arial, 12 point, bold, use the string: <Ft Arial;Pt 120;Bd 1>.

The following table lists the special strings that are available for WSWin Text:

<St s$>	Style. Specify a paragraph style name.
<Ft s$>	Font. Specify a font name. If no name is specified, the font reverts to the font defined in the paragraph style.
<Pt n%>	Point size. Specify the value in decipoints. If n% = 0, the size reverts to the size defined in the paragraph style.
<Bd n%>	Bold. Specify 1 to turn bold on or 0 to turn bold off.
<It n%>	Italics. Specify 1 to turn italics on or 0 to turn italics off.
<Un n%>	Underline. Specify 1 for single underline on, 2 for double underline on, or 0 to turn underline off.
<Ss n%>	Superscript/subscript. Specify 1 for superscript on, 0 for superscript off, 2 for subscript on, or -1 for subscript off.
<Tr n%>	Track kerning. Specify the track kerning value in 1/1000s of an em. A positive value spaces the letters farther apart; a negative value spaces them closer together. An em is the same width as the point size. To tighten characters by 10%, use <Tr -100> (100/1000 is .10, or 10%). If n% = 0, the text returns to the track kerning specified in the paragraph style.
<Kr n%>	Pair kerning. Specify the pair kerning value in 1/1000s of an em. The same rules apply as for track kerning.
<Nb>	Nonbreaking hyphen.
<Dh>	Optional (discretionary) hyphen.
<So n%>	Strikeout. Specify 1 to turn strikeout on or 0 to turn it off.
<Tb>	Tab character.
<Sp n%>	Fixed space. Specify 0 for en space, 1 for em space, 2 for figure space, or 3 for thin space.
<Cr>	Carriage Return.
<Cb>	Column break.
<Pb>	Page break.
<Ts>	Thousands separator.
<Ds>	Decimal separator.
<Cl n%>	Color. Specify the color value as a 32-bit integer. To specify a return to the original paragraph style color, set n% = RGB(255, 255, 255) + 1 (16777216).

The following example shows a WSWin text string:

```
"<St BodyText>Now is the <Bd 1>time for<Bd 0> <It 1>all<It 0> good people
   to come to the aid of their <Ul 1>country.<Ul 0;Cr;St BodyText>"
```

Notice that the special tags at the end of the string were combined using semicolons.

Functions vs. Subroutines

The WSWin macro language has two types of macro instructions - functions and subroutines.

Functions return a value that can be either assigned to a variable or compared with other information. Functions that return a string must have a dollar sign ($) for the last character of the function name. Functions that return a numeric value do not need the percent sign. Functions use parentheses to surround the parameters passed to the function. In the following example, the LEFT$ function returns a string containing the specified number of characters from the beginning of another string:

```
leftString$ = LEFT$("this is a string", 7)
```

When this function is executed, the variable **leftString$** will contain "this is". Notice that both parameters passed to the function were literal values. They could also be variables that were previously assigned the values shown.

Subroutines do not return a value, but perform an action. You can use subroutines to change the values of variables, but the variables are always passed to the subroutine as parameters. For example, the following subroutine gets the current paragraph style name and causes the current paragraph to use the global style:

```
GetStyle styleName$, local%
```

The subroutine places the name of the current paragraph style in the **styleName$** variable and sets the **local%** variable to 1 for local changes (such as an indent setting different from the global paragraph style).

StarBASIC also has statements that control the flow of the macro or describe a dialog box. All the subroutines and functions in the WSWin macro language are described in the Macro Language Reference section.

Macro System Is Not Case Sensitive

When a macro is compiled, the names of all variables, functions, and subroutines are converted to uppercase. Therefore, you cannot have a variable called Left$, for example, because when converted to uppercase it would conflict with the StarBASIC function called LEFT$.

Literal strings surrounded by double quotes are not converted when the macro is compiled.

Macro Comments

You can use comments in a macro to describe how the macro works. If you plan on sharing your macros with other users, include comments in the macro to explain your techniques. Comments also make a macro easier to modify.

To include a comment in a macro, use the REM (for "remark") statement. The comment must stand alone on the line, and the REM must be at the beginning of the comment (spaces and tabs are ignored at the beginning of a line).

To include a comment on a line that also contains a macro instruction, type an apostrophe (') after the instruction and type the comment after the apostrophe. The macro compiler will ignore any characters to the right of an apostrophe until it reaches the end of the line. The following example shows comments on every macro line:

```
CharRight 1, 0,       ' Move one character to the right
SentenceUp 1, 0       ' Go to beginning of sentence
SentenceDown 1, 1     ' Select to the end of sentence
```

To have a macro description appear in the WSWin Status Bar when a menu item or Toolbox button assigned to the macro is highlighted, make sure the first line of the macro contains the following comment:

```
REM Description: This is the macro description.
```

(Use your own description text.) If the first line of the macro is not the description, a default description will be used. Keep the description brief so that it fits on the Status Bar.

When you edit or save a recording, WSWin adds the description line in the macro for you.

Formatting a Macro for Readability

Macros that are formatted are easier to follow. Formatting also helps you to find and fix errors in the macro.

For example, use tabs to indent lines between flow control statements such as IF...ENDIF and FOR...NEXT. To separate the macro in sections, add blank lines.

You can use uppercase, initial caps, and lowercase to set apart StarBASIC statements, WSWin statements, and your own variables. Start variables with a lowercase letter. If a variable reads like more than one word, capitalize the first letter the second and subsequent words, such as styleName$, pointSize%, and frameWidth%.

Improving Macro Running Speed

Running a macro that performs many actions that modify a document, such as changing paragraph styles or inserting text, can take time to complete. You can speed up the macro by freezing the screen with the ViewFreezeScreen function.

While the screen is frozen, changes in the document do not appear on screen and the macro executes more quickly.

Keep the following tips in mind when you use the ViewFreezeScreen function:

- Place the ViewFreezeScreen function after any macro statements that display dialog boxes or input boxes.
- Before the macro exits, restore the screen using ViewFreezeScreen(0). If you forget to restore the screen, WSWin might exhibit strange behavior.
- When you debug a macro, place REM before all ViewFreezeScreen statements, so you can see the results of the macro in the document. You can remove the REMs when your macro is debugged.

Planning a Macro

It is a good idea to plan the macro before you begin programming. Keep the following criteria in the mind:

- Consider these things as you plan your macros. How much of any given process do you want to automate? Will the macro be specific to a particular document and not much use in another? What input boxes and dialog boxes are needed to obtain information while the macro is running?
- If you're writing a macro that requires the insertion point to be in a certain location, keep in mind that the ways to move the insertion point around the screen. In one document, for example, you might press the Home key and then the Right Arrow key five times to move to the beginning of the second word in the line. In another document that might not work. It would be better to move the insertion point word by word.
- Building error checking in your macros prevents unexpected results. For example, you can tell a macro to terminate if the active frame isn't suitable for the macro, such as a graphic frame when a text frame is required. You can also keep the user from entering inappropriate information, such as an extremely large value for the line height.

Related Topics

Designing the Macro

Designing the Macro

Take some time to lay out the steps a macro will perform. Use plain English to describe the problem to be solved and the macro solution.

Use the Macro Editor to keep your notes makes it easier to write the macro from the steps. If you put REM (or an apostrophe) at the beginning of each step, then the steps become the comments for the macro.

For example, you want to write a macro to insert either an opening or closing double quote (the kind used by typographers) depending on whether the insertion point is at the beginning or end of a word. First, state the problem and then write the steps, as shown below.

```
REM Description: Insert typographical double quote
REM Problem: we want to insert an opening double quote
REM if the insertion point is at the beginning of a
REM word.  If the insertion point is at the end of
REM a word (or in the middle), we want to insert a
REM closing double quote.
REM The steps to follow are:
REM Find out the character before the insertion point.
REM If the character is a space, we're at the start of a word.
REM If the character is a carriage return, we're also at the
REM    start of a word.
REM If the character is a tab, we're also at the start of a word.
REM If we're at the start of a word, insert an opening double
REM    quote.
REM Otherwise, insert a closing double quote.
```

Next, you can fill out the macro with the macro instructions to carry out the steps you have described. Keep it simple. For example, you might want to limit error checking until you make sure that the basic steps work correctly. Then you can go back and add the appropriate error checking statements to make sure your macro performs exactly as you expect.

The next section shows the macro described above, with the comments included. Additional comments are added with apostrophes so you can see how the macro actually works.

Related Topics

Macro Example

Example

```
REM Description: Insert typographical double quote

REM Problem: we want to insert an opening double quote
REM if the insertion point is at the beginning of a
REM word.  If the insertion point is at the end of
REM a word (or in the middle), we want to insert a
REM closing double quote.

REM The steps to follow are:

REM Find out the character before the insertion point.

prevChar$ = GetPrevChar$(0)

   ' The GetPrevChar$ function returns the character to the
   ' left of the insertion point.  We are storing it in a string
   ' variable so we can use it again later.  The parameter specifies
   ' that we don't want to look at tags, like Bold or Font.

REM If the character is a space, we're at the start of a word.

IF prevChar$ = " " OR LEFT$(prevChar$, 3) = "<Sp" THEN startWord% = 1

   ' We'll use the numeric variable startWord% to keep track
   ' of whether we are at the start of a word or not.  If the
   ' previous character is a regular space (shown as a space
   ' between two double quotes) or a special space, like the
   ' figure space, thin space, en space, or em space, then we
   ' must be at the beginning of a word.  All the special space
   ' characters are represented by "<Sp" followed by a number
   ' and a closing angle bracket.  We just need to look at the
   ' first three characters to see if it is one of them.

REM If the character is a carriage return, we're also at the
REM    start of a word.

IF prevChar$ = "<Cr>" THEN startWord% = 1

   ' The GetPrevChar$ function returns this string if the
   ' previous character is a carriage return.  Because we told
   ' the function to ignore tags, it doesn't check for the
   ' style tag.

REM If the character is a tab, we're also at the start of a word.

IF prevChar$ = "<Tb>" THEN startWord% = 1

   ' The GetPrevChar$ function returns this string if the
   ' previous character is a tab.  Tabs are sometimes
   ' used at the beginning of a paragraph, before the first
   ' word.

REM If we're at the start of a word, insert an opening double
REM    quote.
```

```
IF startWord% = 1 THEN
   TypeText CHR$(147)

   ' If our variable is equal to 1, then we are at the
   ' beginning of the word.  The TypeText subroutine is used
   ' to insert text in the document.  The CHR$ function is
   ' a StarBASIC function that returns a single-character
   ' string representing the character in the ANSI character set
   ' at the specified position.  The opening double quote is
   ' ANSI character number 147.

REM Otherwise, insert a closing double quote.

ELSE
   TypeText CHR$(148)
ENDIF

   ' All numeric variables begin with a value of 0 when they
   ' are created by the macro system.  So we know that if none
   ' of the IF subroutines was true, then the variable startWord%
   ' must still be 0.  So the ELSE statement is executed.
   ' Notice how the lines between the IF, ELSE, and ENDIF statements
   ' were indented.  That makes them easier to read and follow.
   ' The closing double quote is ANSI character 148.
```

Although this macro works, it isn't as efficient as it could be. To see how the macro can be streamlined, look at the DBLQUOTE macro in the macro directory.

Related Topics

Planning a Macro

Advanced Techniques

Using Dialog Boxes and the Dialog Editor

Many times you need to get information from the user before your macro performs any action. For simple information, you can use the built-in INPUTBOX$ function to get a string from the user. If you want more control over the information, such as with a list of choices, you can use a dialog box.

To create a new dialog box in a macro, put the insertion point where you want the dialog box to be inserted and choose Edit▸Dialog box. The Dialog Editor main window appears and provides a blank dialog box. Add the dialog box controls using the Dialog Editor tools. When you exit the Dialog Editor, the dialog box definition is placed in the macro at the insertion point.
Once the dialog box statements are in the macro, you can adjust the sizing of controls, variable names, etc. You should also make sure that the order of the controls is correct, so that the user can tab from control to control in a natural order.

To see sample macros that use dialog boxes, look at any of these WSWin macros: BULLETS, CNT, LSTFILES, PAGEBRDR, PAGENUM, AND QUIKPATH.

Related Topics

Dialog Editor Tools

Dialog Editor Tips

Dialog Editor Tools

To add controls to the dialog box, click a tool on the Toolbox or choose one of the tools from the Tools menu. The tools are described below:

- The selection tool (arrow) is for selecting controls in the dialog box. As soon as you create a tool, the selection arrow is reactivated.
- The text tool (capital T) is for static text controls. Use static text for labels of other controls or for instructions in the macro.
- The edit box tool is for creating edit box controls. Edit boxes always return a string. If you need a number, convert the string to a numeric value .
- The option (or radio) button tool is for creating radio buttons. Option buttons are grouped using the OPTIONGROUP statement. Any option buttons that are after an OPTIONGROUP statement and before any other OPTIONGROUP statements will be treated as a group. The Dialog Editor will try to group the option buttons correctly as you create them.
- The checkbox tool is for creating checkboxes. Checkboxes set a numeric variable to 1 if they are checked or to 0 if they are not checked.
- The combobox tool is for creating comboboxes and dropboxes. A combobox is an edit box attached to a list. The user can type any string in the edit box or choose an item from the list. A dropbox is like a combobox, except the list is only displayed if you click the button next to the edit box.
- The list tool is for creating lists and directory boxes. A list restricts the user to selecting only from the items in the list. A directory box is a combobox that displays files and/or directories in the list beneath the edit box.
- The groupbox tool is for creating groupboxes. A group box doesn't get any input, but it helps to arrange controls to make the dialog box easier to understand.
- The pushbutton tool is for creating regular pushbuttons. You can give a regular pushbutton any string label to display on the button. If the button is pressed, its assigned number is returned by the DIALOG function. Pushbuttons always close the dialog box. The Dialog Editor assigns a number to each pushbutton you create, starting with 3 (1 and 2 are reserved for the OK and Cancel buttons).
- The OK button tool is for creating the OK button. Only one OK button is allowed in a dialog box.
- The Cancel button tool is for creating the Cancel button. Only one Cancel button is allowed in a dialog box.

Dialog Editor Tips

Creating Dialog Box Controls

Sizing Dialog Box Controls

Aligning Dialog Box Controls

Sizing Dialog Boxes

Creating Dialog Box Controls

To create more then one of the same type of control, double-click the tool on the Toolbox to lock the tool. When a tool is locked, it stays pressed so you can create more of the same kind of control. That makes it easy to create three option buttons together.

To resize a control, select the control by clicking it and grab an edge . Resize the control the same way you resize a document window. If no control is selected, you can resize the dialog box itself.

You can also resize a control by double-clicking on the control, or by pressing Enter when the tool is selected. In the dialog box, you can specify the precise values for the location and size of the control. If more than one control is selected, or if no control is selected, you can change the size of the dialog box. You can also change the text of the control, the name of the variable associated with the control, or for comboboxes and dropboxes, the type of control.

Related Topics

Sizing Dialog Box Controls

Aligning Dialog Box Controls

Sizing Dialog Boxes

Sizing Dialog Box Controls

- Static text controls are normally 8 units tall for a single line. Multiply the number of lines times 8 if the text is longer than one line.
- Edit boxes are normally 12 units tall. If the edit box will hold more than one line of text, multiply the number of lines times 8 and add 4 to the result (a three-line edit box would be 28 units tall - 3 x 8 = 24 + 4 = 28).
- Option buttons and checkboxes are 10 units tall. When grouping option buttons or checkboxes, you can arrange them vertically so that they are next to each other. For example, if the first option button has a Y value of 10, then next one can be at 20, the next at 30, and so on.
- A list should be sized the same way as a multiline edit box, multiplying the number of lines times 8, and adding 4 to the result. A list that displays 10 items would be 84 units (10 x 8 = 80 + 4 = 84).
- To calculate the correct size of a combobox or dropbox, calculate the size of the list and add 12 for the attached edit box. A combobox with 10 items would be 96 units (84 + 12 = 96).
- If any control has a static text label, add 10 to the Y position of the label to determine the position of the control. For example, if an edit box has a label above it with a Y position of 4, then the edit box should have a Y position of 14.
- Allow a minimum of 4 units between the edge of dialog box and the controls inside the dialog box.
- Separate groups of controls by 6 units both horizontally and vertically. Controls within the group can be separated by 2 units, except for option buttons and checkboxes, which do not need any separation.

Related Topics

Creating Dialog Box Controls

Aligning Dialog Box Controls

Sizing Dialog Boxes

Aligning Dialog Box Controls

You can use the alignment and distribution controls to make the finished dialog box more attractive. Select the controls you want to modify. To select more than one control, draw a selection box around all the controls or hold the Shift key down and click each one. (Working with dialog boxes is similar to working with graphic objects in a WSWin document graphic frame.)

When you have selected the controls, click the appropriate buttons in the Align/Distribute Toolbox or choose the commands from the Arrange menu. For example:

- To align the selected controls along their left edges, select the Align Left button.
- Once they are aligned, you can drag the right edge of the selection box to make it wider or narrower. Then click the Same Width button (the last button in the top row of the Toolbox). The controls are now the same width as the selection box.
- Drag the bottom edge of the selection box to make it taller or shorter. Then click the Distribute Top-Bottom (the fourth button in the top row of the Toolbox) so that they are evenly spaced within the selection box.
- While the controls are selected together, you can drag them to a precise position.

You can use the buttons in the top row of the Align/Distribute Toolbox to arrange controls in a single column. If you have controls arranged in a row from right to left, use the buttons on the bottom row of the Toolbox.

Related Topics

Creating Dialog Box Controls
Sizing Dialog Box Controls
Sizing Dialog Boxes

Sizing Dialog Boxes

If you do not specify the X and Y position of the dialog box when you resize it, it will automatically be centered on screen when the macro displays it.

The numbers used for specifying the position and size of the dialog box and controls represent dialog box units. Dialog box units are based on the size of the current system font. Dialog boxes display correctly no matter what display driver or system font you use. When sizing a dialog box and its controls, make sure the dialog box is not larger than 300 units wide or 200 units tall. If you plan on sharing your macros with others, check the dialog box at different resolutions and font sizes to determine the best size.

Related Topics

Creating Dialog Box Controls

Sizing Dialog Box Controls

Aligning Dialog Box Controls

Writing Custom Functions and Subroutines

If you have a group of instructions that will be repeated in a macro, create a custom function or subroutine for those instructions. The instructions are written once in the macro, and can be called from different places within the macro. If the instructions are changed, the changes take effect everywhere.

To create a custom function or subroutine, first you must declare it. This tells the macro system that the name you give will be a function or subroutine, instead of a variable. To declare a function, use the following statement at the beginning of your macro:

```
DECLARE FUNCTION CustomFunction%(param1$, param2%)
```

In the example, the first parameter is a string and the second parameter is a number. The function returns a number.

To declare a subroutine, use the DECLARE SUB statement, as shown in the example below.

```
DECLARE SUB CustomSubroutine(param1$, param2%)
```

Although you do not use parentheses when you call a subroutine in the macro itself, you must use them when you declare the subroutine.

After declaring the functions and subroutines, write the main body of the macro. Use the standard syntax for functions and subroutines when you call custom functions and subroutines in the macro. The code for the custom functions and subroutines is placed after the last line of the main body of the macro.

The following example shows a simple macro that uses both a custom function and a custom subroutine.

```
REM Description: A macro that demonstrates custom functions
' First, the custom function and subroutine must be declared.
DECLARE FUNCTION CustomFunction% (string$, int%)
DECLARE SUB CustomSubroutine (string$, int%)

' This is the main body of the macro.  First, call
' CustomSubroutine to assign values to the two variables.
CustomSubroutine str$, num%
' Now call CustomFunction to perform the operation on the
' two variables and return the value.
ret% = CustomFunction%(str$, num%)
' This is the end of the main body of the macro.  Now we
' can add the code for the custom routines.

' CustomFunction - takes a string parameter and a number
' The function multiplies the ANSI value of the first character
' of the string times the number and returns the result.
FUNCTION CustomFunction% (string$, int%)
   ' To specify the return value, use the name of the
   ' function as if it were a variable.  By assigning
   ' the result of the calculation to the function
   ' name, the macro system knows what the return value
   ' should be.
   CustomFunction% = ASC(string$) * int%
END FUNCTION
' The END FUNCTION statement tells the macro system that this
' is the end of the function.  The macro then continues to
' run at the next statement after the function call.

' CustomSubroutine - takes a string parameter and a number.
' The subroutine assigns a literal string to the string parameter
' and a numeric value to the number variable.
SUB CustomSubroutine (string$, int%)
```

```
    ' Assigning the string to the parameter actually
    ' modifies the variable that was passed to the subroutine.
    string$ = "Test one"
    ' The same is true for the numeric variable.
    int% = 10
END SUB
' The END SUB statement, like END FUNCTION, tells the macro
' system to return to the calling routine.
```

Notice that CustomSubroutine modified the two variables that were passed to it. This gives custom functions and subroutines more power and flexibility.

Related Topics

Using Dynamic Link Libraries

Using Dynamic Link Libraries

You can use functions that are built in Windows to get information about the operating environment, as well as public functions in other dynamic link libraries (DLLs) in other programs.

To find out how to use functions in DLLs, you need specific technical reference material. For example, to use the Windows DLLs, you need the Windows Software Development Kit.

Before you can use a DLL function or subroutine, you must declare the function. Use the following statements:

```
DECLARE FUNCTION GetActiveWindow LIB "user.exe" () AS WORD
DECLARE FUNCTION WinExec LIB "kernel.exe" (path AS STRING, show AS WORD)
   AS WORD
DECLARE SUB CloseWindow LIB "user.exe" (win AS WORD)
```

The LIB keyword tells the macro system that the function or subroutine is in the specified DLL. Make sure you capitalizate the function or subroutine name as specified in the technical reference for the DLL.

You can use percent (%) and dollar sign ($) indicators instead of the AS keyword for parameters, but the AS keyword has more flexibility and supports additional variable types that are not part of the regular WSWin macro system.

Note WSWin numeric variables are long integers and cannot be used interchangeably with integer (or WORD) parameters. To convert a long integer to a word or integer parameter, use the I2W function. To convert the return value of a function that returns a word back to a long integer, use the W2I function. These are described in the Macro Language Reference section.

For more information on using DLL functions, look at any of these WSWin macros: CHARMAP, CALC, PBRUSH, AND WSWAHD3.

Using Automatic Macros

WSWin can run a macro when you call certain commands on the File menu. These auto macros enhance the way the commands work. For example, you can create a macro that runs when you open a file to tile the windows of all open documents.

There are two ways you can have automatic macros. The easiest way is to give the macro file a special name. The following list describes the filenames for the automatic macros:

ONSTART	Runs when WSWin is first started
ONNEW	Runs when the New Defaults button on the Toolbox is chosen
ONNEWDLG	Runs when the File▸New command is called
ONOPEN	Runs when the File▸Open command is called
ONCLOSE	Runs when the File▸Close command is called
ONPRINT	Runs when the File▸Print command is called
ONSAVE	Runs when the File▸Save command is called
ONSAVEAS	Runs when the File▸Save As command is called
ONREVERT	Runs when the File▸Revert to Saved command is called
ONEXIT	Runs when the File▸Exit command is called

The other way to specify auto macros is to modify the WSW.INI file to tell WSWin what macro to run. Open the WSW.INI file in a text editor such as the Windows Notepad. Look for the section titled [Auto Macros]. If the section isn't found, you can add it yourself. You might want to add the section so that it looks like this:

```
[Auto Macros]
OnStart=
OnNew=
OnNewDlg=
OnOpen=
OnClose=
OnPrint=
OnSave=
OnSaveAs=
OnRevert=
OnExit=
```

If there is no macro name after the equal sign, WSWin will try to run the auto macro with the default name. For example, when you choose the File▸Open command, WSWin will try to run the macro named ONOPEN in the default macro directory. To run a macro called TILE when you choose the File▸Open command, change the entry in the WSW.INI file as follows:

```
OnOpen=tile.wmc
```

Note Auto macros replace the functionality of the original command. If you write a macro for OnOpen, you must execute the FileOpen command within your macro, if you want the command to open a file. The same is true for the OnStart macro; it must be written to handle a situation where a filename is sent to WSWin on the command line. See the LOADLAST macro for an example .

Debugging Macros

Macros are hardly every perfect the first time they are run. Incorrect error handling and errors in logic can prevent a macro from executing correctly. Two different types of errors can occur: compiler errors and run-time errors.

To find compiler errors, try to run a macro from the Macro Editor by choosing Debug▸Run or check the syntax by choosing Debug
▸Check Syntax. If the macro compiler finds errors, a separate window appears below the macro window listing the errors.
Other errors can cause a macro to fail when it runs. For example, you might misspell a variable name in one place in the macro, causing an invalid value to be passed to a function. You might execute a statement that is invalid in the current situation (such as when WSWin is in frame mode, but edit mode is expected).

Related Topics

How to Debug Macros

Common Programming Errors

How to Debug Macros

The Macro Editor has ways to make debugging easier. If the compiler reports errors, you can double-click an error message in the error window and go directly to the line where the error occurred. The insertion point will be placed at the beginning of the line.

If the compiler reports no errors, you can use the Debug▸Step and Debug▸Trace commands to watch individual statements being executed. Use Step when you want to step through your custom functions and subroutines. Use Trace when you want to stay in the main body of the macro.

If you don't want to watch every statement, you can use Debug▸Toggle Breakpoint to set a breakpoint at the line where you want the macro to pause. Once the macro is paused, you can use Step and Trace to continue executing or you can set new breakpoints.

You can also use the Debug▸Add Watch and Debug▸Quick Watch commands to look at the contents of individual variables.

Common Programming Errors

When writing macros, most users run in one or more of the following problems:

- Cascading Error Messages
- Beginning Statements Without Required Closing Statements
- Missing Parentheses
- End of Line Errors
- Bad File Names
- Replacement Errors
- Mode Errors

Related Topics

Other Errors

Cascading Error Messages

Cascading error messages are a group of error messages that occur after compilation. Typically, if you fix the problem associated with the first messages, the others will be fixed as well. For example, if your first message is

```
error at <EOL> deleting <EOL>
```

you've probably omitted something required on the previous line. Fix the error and check the syntax again. You'll probably eliminate many errors with one fix.

No Matching Statements

Some macro statements require a matching statement later in the macro. For example, to define a style, you need a DefineStyle statement and an EndDefineStyle statement. This is also true for the DefineFrameStyle statement, as well as for FOR...NEXT, DO...LOOP, IF...ENDIF, and WHILE...WEND statements. The error message tells you which portion of the statement set you've omitted.

Missing Parentheses

Missing parentheses can cause a number of problems. If you omit them from a function, for example, the compiler assumes the keyword you've entered is a variable. If you forget to put them in when declaring a subroutine or a function, you can get cascading errors that don't describe the problem. Missing parentheses can also cause an end of line error.

End of Line Errors

End of line errors typically occur when you omit something from the previous line. For example, if you omit the parentheses when you call a function, the macro compiler displays an end of line error.

Bad File Names

The compiler doesn't give error messages for bad file names. It's your responsibility to type the name correctly using the characters allowable in DOS and the right syntax.

Replacement Errors

When the compiler expects something other than what you've typed, it may indicate that it's replacing what you've typed. For example, if you type

```
i% = 1 TO 10
```

the compiler will display the following message:

```
error at TO replacing with XOR
```

In this case, TO doesn't work, and XOR is the compiler's next choice in its grammar.

Mode Errors

Mode errors occur only when a macro is running. If you are in the draft editor and the macro tries to create a frame, WSWin will report an error. If you are in frame mode and the macro tries to insert text, an error will result. To correct these errors, check for the correct mode and add statements to change to the desired mode before the macro continues.

Other Errors

You should also be aware of the following error conditions that may occur:

- **Lengthy Statements** If a statement contains more than 512 characters, compilation errors may display the incorrect line number.
- **Numerous Errors** The compiler reports a maximum of 11 errors. Once the 11th error has been found, the compiler stops. You must then correct the displayed errors and check the syntax again.
- **Memory Allocation** Memory allocation errors may result if you use very large, multi-dimensional arrays or deeply nested custom procedures or functions.

Related Topics

Common Programming Errors

Macro Language Reference

StarBASIC Subroutines and Functions

- Variable and Array Statements
- Operators
- Flow Control Statements
- String Statements
- DOS and File Statements
- User Interface Statements

WSWin Subroutines and Functions

- File Statements
- Edit Statements
- View Statements
- Insert Statements
- Style Statements
- Frame Statements
- Table Statements
- Tools Statements
- Information Statements
- Navigation Statements

Variable and Array Statements

StarBASIC supports the following variable and array statements:

- CONST
- DIM
- LBOUND
- LET
- SHARED
- STATIC
- UBOUND

The following keywords are reserved for future use and cannot be used as subroutine names or line labels: DATA, DEFINT, DEFSNG, DEFDBL, DEFLNG, DEFSTR, TYPE.

CONST

Description

Declares symbolic constants for use in place of numeric or string values.

Syntax

CONST constant{$ | %} = expression [, constant{$ | %} = expression, ...]

Parameters

constant{$ | %} - Follows the same rules as for StarBASIC variable names. A type-declaration character must follow the name ($ for strings, % for integers).

expression - An expression consisting of literals (such as 1), other constants, and any of the arithmetic and logical operators.

Example

```
CONST BLANK% = 32, LF% = 10
```

Declares symbolic constants for the ASCII values of nonprinting characters blank and line feed.

DIM

Description

Specifies the dimensions, or number of elements, in an array. The dimensions must be literal or defined constants, not variables. The number of dimensions in any array should not exceed 5.

If you do not use a TO clause to specify the range of elements, the default (1 to n) is used.

Syntax

DIM arrayvar{$ | %} ({upperbound | lowerbound TO upperbound} [, ...]) [,...]

Parameters

arrayvar{$ | %} - Name of the array (must include a type-declaration character).

upperbound - The upper bound of the array.

lowerbound - The lower bound of the array.

Example

```
DIM salesdata$(10)
```

Creates a string array named salesdata that consists of 10 elements.

```
DIM salespeople%(-5 TO +7)
```

Creates an integer array named salespeople that consists of 13 elements.

See Also

LBOUND, UBOUND

LBOUND

Description

Returns the lower bound for the indicated dimension of an array. If the dimension is omitted, the lower bound of the first dimension is returned.

Syntax

retval% = LBOUND(array{$ | %}[,dimension])

Parameters

array{$ | %} - The array being dimensioned.

dimension - An integer variable or numeric constant ranging from 1 to the number of dimensions in array. Indicates which dimension's lower bound is returned; if omitted, the limit of the first dimension is returned.

Returns

The lower bound for the indicated dimension of an array.

Example

```
DIM a%(-5 TO 7, 10)
i% = LBOUND(a%,1)
i% = LBOUND(a%,2)
```

Sets i% to -5 and 1, respectively.

See Also

DIM, UBOUND

LET

Description

Assigns the value of an expression to a variable. The keyword is optional.

Syntax

[LET] variable{$ | %} = expression

Parameters

variable{$ | %} - Receives the value of expression.

expression - The expression (either literal or calculated) to be assigned to the variable. It must be the same class as variable (if variable is a string, expression must be a string; if variable is a number, expression must be a number).

Example

```
LET stringVar$ = "This is a string."
```

Assigns the string "This is a string." to the variable stringVar$.

```
result% = (a% + b%) / c%
```

Assigns the result of the sum of the values of variables a% and b%, divided by the value of c%, to the variable result%. The LET keyword is omitted.

SHARED

Description

Lets you override the default and make specified variables available to both the local procedure and the main program. By default, subprogram variables are local to those procedures and cannot be accessed by other procedures or the main program. Using the SHARED keyword allows a variable to be available to a subprogram without being passed as a parameter.

Syntax

SHARED var{$ | %} [,var{$ | %}...]

Parameters

var{$ | %} - Any valid variable.

Example

```
REM (Main Program)
DECLARE SUB a(a%)
DECLARE SUB b(b%)
SHARED i%
i% = 77
CALL a(a%)
CALL b(b%)
REM(Subroutine A)
SUB a(a%)
a% = i%
END SUB
REM (Subroutine B)
SUB b(b%)
b% = i%
END SUB
```

Because the variable i% is shared in the main program, before the subroutines are called, the subroutines have access to the value of i% without requiring it as a passed parameter. When this macro finishes running, both a% and b% will have the same value as i%. Variables must always be shared before they are used, or run-time errors can result.

See Also

DIM, FUNCTION, SUB

STATIC

Description

Preserves contents of selected variables, letting them retain their value between calls rather than being re-created each time.

Syntax

STATIC var{% | $}

Parameters

var{% | $} - Any valid variable.

Example

```
REM (Main Program)
DECLARE FUNCTION staticFunc% (a%)
FOR i% = 1 to 5
   j% = staticFunc%(i%)
NEXT i%
REM (Static Function Example)
FUNCTION staticFunc%(a%)
STATIC staticVar%
' Because staticVar% is STATIC, it retains its previous value
' each time the function is called
staticVar% = staticVar% + a%
' The function returns the current value of staticVar%
staticFunc% = staticVar%
END SUB
```

The variable staticVar% in the function is created as a STATIC variable, so that its value remains unchanged each time the function is called. In the main program, a FOR loop calls the function five times. In each of the fime function calls, the results are as follows:

1 The first time the macro runs, staticVar% has a value of 0 because it is created for the first time. The passed parameter , i%, has a value of 1, and the variable also has a value of 1.

2 In the second call, staticVar% has a value of 1 and the passed parameter has a value of 2. So the calculation causes staticVar% to be 3.

3 In the third call, staticVar% is equal to 3 and i% is equal to 3, so staticVar% has a new value of 6.

4 In the fourth call, staticVar% is 6 and i% is 4, giving staticVar% a new value of 10.

5 In the last call, staticVar% is 10 and i% is 5, giving staticVar% a value of 15.

See Also

DIM, FUNCTION, SHARED, SUB

UBOUND

Description

Returns the upper bound for the indicated dimension of an array. If dimension is omitted, the upper bound of the first dimension is returned.

Syntax

retval% = UBOUND(array{$ | %}[,dimension])

Parameters

array{$ | %} - The array being dimensioned.

dimension - An integer variable or numeric constant ranging from 1 to the number of dimensions in array: indicates which dimension's lower bound is returned; if omitted, the limit of the first dimension is returned.

Returns

The upper bound for the indicated dimension of an array.

Example

```
DIM a%(-5 TO 7, 10)
i% = UBOUND(a%,1)
i% = UBOUND(a%,2)
```

Sets i% to 7 and 10, respectively.

See Also

DIM, LBOUND

Operators

StarBASIC supports the following operators:

Arithmetic Operators: + (Addition), - (Subtraction), * (Multiplication), / (Division), ^ (Exponentiation), MOD (Modulus).

Relational Operators: =, <>, >, <, >= (=>), and <= (=<).

Logical Operators: AND, OR , XOR, NOT.

Numeric Conversion Functions: I2W, W2I.

Arithmetic Operators

Description

Use arithmetic operators to perform calculations on numbers and numeric variables. The addition operator (+) combines strings together. The subtraction (-) operator also converts the sign of a variable.

Arithmetic operations are performed according to an operational hierarchy. The following operators take precedence over other operators in an expression:

^ (Exponentiation) - These operations are always executed first.

* (Multiplication), / (Division), MOD - These operations are always executed second, in order from left to right.

+ (Addition), - (Subtraction) - These operations are always executed last.

To control the order of execution, use parentheses to group subexpressions together.

Syntax

retval% = a% + b% - c% * d% / e% ^ f%

Parameters

Place each arithmetic operator between two expressions.

Returns

The result of the calculation.

Example

```
i% = a% + b%
```

Assigns the sum of the values of variables a% and b% to the variable i%.

```
j% = j% - 2
```

Decreases the variable j% by 2.

```
j% = -j%
```

If j% was greater than 0, converts the value of j% to a negative number. If j% was less than 0, j% becomes a positive number. This is the same as the expression "j% = 0 - j%".

```
k% = (c% + d%) * 5
```

Adds the values of variables c% and d% together, multiplies the sum by 5, and assigns the result to the variable k%. Note that the parentheses force the calculation of the addition before the multiplication.

```
m% = e% / f%
```

Assigns the result of the value of variable e% divided by the value of f% to the variable m%. Because StarBASIC numeric variables are always whole numbers, any fraction that results from the division is lost. The result is not rounded.

```
n% = g% ^ 2
```

Assigns the result of the value of g% raised to the second power to the variable n%. The expression "g% ^ 2" is the same as "g% * g%".

```
firstString$ = "test 1"
secondString$ = "test 2"
result$ = firstString$ + " " + secondString$
```

Assigns the value of firstString$ followed by the value of secondString$ to the variable result$ with a space in between. In this example, result$ contains "test 1 test 2".

See Also

MOD, AND, OR , XOR, NOT

Relational Operators

Description

The relational operators are used to compare two operands.

Syntax

IF op1{% | $} = op2{% | $} THEN ...

IF op1{% | $} <> op2{% | $} THEN ...

IF op1{% | $} > op2{% | $} THEN ...

IF op1{% | $} < op2{% | $} THEN ...

IF op1{% | $} >= op2{% | $} THEN ...

IF op1{% | $} <= op2{% | $} THEN ...

Parameters

op1{% | $}, op2{% | $} - Operands. The operands can be either integers or strings, but both operands must be the same type.

Returns

TRUE if the expression is true.

Example

```
IF a% = b% THEN BEEP
```

The computer beeps if both a% and b% hold the same value. Because the equal sign (=) is used for both assignment and comparison, StarBASIC will recognize the comparison operator only when it is within an IF, DO...LOOP, or WHILE...WEND statement.

```
IF str1$ <> str2$ THEN BEEP
```

The computer beeps if the values of the two string variables are not equal. Strings are always compared in their entirety, and case is significant. So if str1$ is "Cat" and str2$ is "cat", a beep will sound, because the strings are different.

```
IF a% > b% THEN BEEP
```

The computer beeps if the value of a% is greater than the value of b%.

```
IF str1$ < str2$ THEN BEEP
```

The computer beeps if the value of str1$ is less than the value of str2$. When comparing strings, the ANSI value of the characters in the strings are compared. So if str1$ is "cat" and str2$ is "dog", a beep will sound, because "c" comes before "d" in the ANSI character set.

```
IF a% >= b% THEN BEEP
```

The computer beeps if a% is greater than or equal to b%.

```
IF a% <= b% THEN BEEP
```

The computer beeps if a% is less than or equal to b%.

Related Topics

Operators

AND

Description

As a logical operator, AND returns a value of TRUE if both operands are true. As an arithmetic operator, compares the corresponding bits of the operand values.

Syntax

IF op1% AND op2% THEN. . .

result% = op1% AND op2%

Parameters

op1%, op2% - Operand, typically a boolean expression.

Returns

TRUE if both operands are true. In StarBASIC, a value of TRUE is any value not equal to 0. FALSE is always 0.

Example

```
IF a% = 0 AND b% = 0 THEN BEEP
```

If the value of a% is equal to 0 and the value of b% is equal to 0, a beep will be heard. Both expressions must be true for the beep to sound.

```
n% = a% AND b%
```

Compares the bits of the variables a% and b% and sets the corresponding bits of n% to 1 if the same bit is 1 in both a% and b%. For example, if a% is 9 and b% is 12, then the bit operations would be performed as follows:

```
0 0 0 0 1 0 0 1       =  9
0 0 0 0 1 1 0 0       = 12
0 0 0 0 1 0 0 0       =  8
```

Because only the 3rd bit (bits are always counted from right to left, starting with 0) is 1 in both operands, only the resulting number has the 3rd bit set to 1.

See Also

OR , XOR, NOT

I2W

Description

Converts a long integer to a short integer (WORD). This function is used with functions and subroutines in external DLLs that expect short integers for passed parameters.

Syntax

short = I2W(long%)

Parameters

long% - An integer variable; all WSWin macro integer variables are long (32-bit) integers.

Returns

A short integer value equivalent to the value of the original variable.

Example

```
DECLARE FUNCTION SendMessage LIB "user.exe" (hwnd AS WORD, msg AS WORD,
   wParam AS WORD, lParam AS INTEGER) AS INTEGER
DECLARE FUNCTION FindWindow LIB "user.exe" (classname AS INTEGER, title AS
   STRING) AS WORD
hwnd% = W2I(FindWindow(0, "WSWin 2.0"))
IF hwnd% > 0 THEN
   ret% = W2I(SendMessage(I2W(hwnd%), I2W(16), I2W(0), 0))
ENDIF
```

Retrieves the number of the WSWin application window and sends a close message (16) to the window. Note that because both Windows functions return a WORD, they must be converted back to long integers to be assigned to macro variables.

See Also

W2I, DECLARE...LIB

MOD

Description

Returns the remainder of division between two integers.

Syntax

result% = op1% MOD op2%

Parameters

op1%, op2% - Integer operands.

Returns

The remainder of the division operation.

Example

```
n% = 12 MOD 9
```

If you divide 12 by 9, the result is 1 with a remainder of 3. The MOD operator returns only the remainder, so n% would be equal to 3. The MOD operator is often used with integer arithmetic to find out what is lost when a division operation is performed. You can use the MOD operation to round the result of division if the remainder is greater than half of the divisor.

```
n% = a% / b%
IF (a% MOD b%) >= (b% / 2) THEN n% = n% + 1
```

In this example, if a% is equal to 20 and b% is equal to 7, the result of the division would be 2, with a remainder of 6. The next statement compares the remainder of the division with half of b% (7 divided by 2 is equal to 3 with a remainder of 1). Because 6 is greater than 3, we add 1 to n%. Because 20 divided by 7 is almost 3, the result of these two statements makes n% equal to 3.

See Also

Arithmetic Operators

NOT

Description

As a logical operator, NOT returns TRUE if the operand is false, and FALSE if the operand is true. As an arithmetic operator, NOT returns the opposite of the corresponding bits of the operand, which is sometimes called "two's-complement."

Syntax

IF NOT(op1%) THEN. . .

result% = NOT(op1%)

Parameters

op1% - Operand, typically a boolean expression.

Returns

TRUE if the operand is false, and FALSE if the operand is true. In StarBASIC, a value of TRUE is any value not equal to 0. FALSE is always 0.

Example

```
IF NOT(a% = 0) THEN BEEP
```

If the value of a% is not equal to 0, a beep sounds.

```
n% = NOT(a%)
```

Compares the bits of the variable a% and sets the corresponding bits of n% to 1 if the same bit is 0 in a%. For example, if a% is 9, then the bit operations would be performed as follows:

```
0 0 0 0 1 0 0 1       =   9
1 1 1 1 0 1 1 0       = -10
```

If a bit is 0 in a%, then the same bit in n% is set to 1, and vice versa. This example uses only eight bits to demonstrate the bit operation. StarBASIC integers are 32 bits long.

See Also

OR , XOR, NOT

OR

Description

As a logical operator, returns a value of TRUE if either operand is true. As an arithmetic operator, compares the corresponding bits of the operand values.

Syntax

IF op1% OR op2% THEN. . .

result% = op1% OR op2%

Parameters

op1%, op2% - Operand, typically a boolean expression.

Returns

TRUE if either operand is true. In StarBASIC, a value of TRUE is any value not equal to 0. FALSE is always 0.

Example

```
IF a% = 0 OR b% = 0 THEN BEEP
```

If either the value of a% is equal to 0 or the value of b% is equal to 0, a beep sounds. Only one of the expressions must be true for the beep to sound.

```
n% = a% OR b%
```

Compares the bits of the variables a% and b%, and sets the corresponding bits of n% to 1 if the same bit is 1 in either a% or b%. For example, if a% is 9 and b% is 12, then the bit operations is performed as follows:

```
0 0 0 0 1 0 0 1      =  9
0 0 0 0 1 1 0 0      = 12
0 0 0 0 1 1 0 1      = 13
```

The 3rd, 2nd, and 0 bits (bits are always counted from right to left, starting with 0), are set to 1 because those bits have a value of 1 in either of the two operands.

See Also

AND, XOR, NOT

W2I

Description

Converts a short integer (WORD) to a long integer. This function is used with functions and subroutines in external DLLs that return short integers.

Syntax

long% = I2W(short)

Parameters

short - a short integer (WORD) variable. Many Windows functions return WORD values.

Returns

A long integer value equivalent to the value of the original variable.

Example

See I2W.

See Also

DECLARE...LIB

XOR

Description

XOR is the shorthand expression of "exclusive or." As a logical operator, XOR returns a value of TRUE if either, but not both, operands are true. As an arithmetic operator, it performs XOR operations on the corresponding bits of the operand values.

Syntax

IF op1% XOR op2% THEN. . .

result% = op1% XOR op2%

Parameters

op1%, op2% - Operand, typically a boolean expression.

Returns

TRUE if the either operand is true. If both operands are TRUE, returns FALSE. If both operands are FALSE, returns FALSE. In StarBASIC, a value of TRUE is any value not equal to 0. FALSE is always 0.

Example

```
IF a% = 0 XOR b% = 0 THEN BEEP
```

If the value of a% is equal to 0 and the value of b% is not equal to 0, or vice versa, a beep sounds. One expression must be true and the other false for the beep to sound.

```
n% = a% XOR b%
```

Compares the bits of the variables a% and b% and sets the corresponding bits of n% to 1 if the same bit is different in a% and b%. For example, if a% is 9 and b% is 12, then the bit operations would be performed as follows:

```
0 0 0 0 1 0 0 1       =  9
0 0 0 0 1 1 0 0       = 12
0 0 0 0 0 1 0 1       =  5
```

Because the 3rd bit (bits are always counted from right to left, starting with 0) is 1 in both operands, the 3rd bit in the result is set to 0. The 2nd and 0 bits are different in the two operands, so those bits are set to 1 in the result.

See Also

AND, OR , NOT

Flow Control Statements

A program generally performs various functions. Flow control statements help you control the program's operation.

For example, you can:

- Execute specific statements only if an expression is true with IF...THEN...ELSE...ENDIF
- Execute different sets of statements depending on the value of a variable with SELECT CASE
- Repeat statements while or until some expression is true with DO...LOOP
- Repeat statements a specific number of times with FOR...NEXT
- Call a subroutine or function with CALL
- Go to a different line in the program with GOTO

The following additional flow control statements are available in StarBASIC:

- DECLARE
- DECLARE...LIB
- END
- FUNCTION
- REM
- STOP
- SUB
- WHILE...WEND

CALL

Description

Executes a named subroutine. If arguments are included, you must enclose them with parentheses if the word CALL is used. An empty argument list indicated by ( ) is not allowed.

Note You must DECLARE all user-defined subroutines before you can CALL them.

Syntax

CALL name [(param1{% | $}[, param2{% | $}...])]

Parameters

name - Nameof called subroutine; not case-sensitive.

param1{% | $}, param2{% | $} - One or more optional arguments (variables, constants, or expressions), separated by commas, that are passed to the subroutine.

Example

```
DECLARE SUB thisprogram (menuItem$, price%)
CALL thisprogram (menuItem$, price%)
```

Executes the user-defined subroutine thisprogram, which includes the parameters menuItem$ and price%. Optionally, you can also execute the subroutine with the following statement:

```
thisprogram menuItem$, price%
```

See Also

DECLARE, SUB

DECLARE

Description

Before a macro can call a user-defined function or subroutine, it must be declared with the DECLARE keyword. This specifies the name of the user-defined procedure and the number and type of arguments passed to it. DECLARE is required for all functions and subroutines.

Syntax

DECLARE FUNCTION name{$ | %} ([params{$ | %}] [,...])

DECLARE SUB name

Parameters

name - Name of the called procedure; must correspond to the name of the procedure in the same macro file. If a function is declared, the name must be followed by a type declaration character (% or $).

params{$ | %} - Variable types that are the procedure parameters. Variable types must be specified with a type-declaration character.

Example

```
DECLARE SUB Verify (S$, I%)
```

Defines a subroutine named Verify and specifies the types of arguments-string and integer-passed to it.

```
DECLARE FUNCTION Center% ( )
```

Defines a function named Center that returns an integer. The ( ) indicate an empty parameter list.

See Also

CALL, FUNCTION, SUB

DECLARE...LIB

Description

Before a macro can call a function or subroutine in an external dynamic link library (DLL), it must be declared with the DECLARE...LIB keyword. This specifies the name of a procedure and the DLL that contains the procedure, and it defines the number and type of arguments passed to it. DECLARE...LIB is required for all external functions and subprograms. Use an AS clause to define variables within the DECLARE...LIB statement.

Consult the technical reference for the DLL before you call any of its functions.

Syntax

DECLARE FUNCTION name{$ | %} LIB file$ ([params{$ | %}] [,...])

DECLARE SUB name{$ | %} LIB file$ ([params{$ | %}] [,...])

Parameters

name - Name of the called procedure; must correspond to the procedure name in the DLL module where it resides; name matching is case-sensitive.

file$ - Name of the DLL file to access.

params% - Variable types that are the procedure parameters. Variables types must be specified with a type-declaration character or AS clause.

AS WORD - Defines a variable as a short integer.

AS INTEGER - Defines a variable as an integer.

AS STRING - Defines a variable as a string.

AS LPWORD - Defines a variable as a long pointer to a short integer, as an integer address rather than its actual value.

Example

```
DECLARE FUNCTION GetActiveWindow LIB "user.exe" ( ) AS INTEGER
DECLARE SUB CloseWindow LIB "user.exe" (win AS INTEGER)
```

The first statement defines a procedure named GetActiveWindow (a Windows function that does not require a type-declaration character). The keyword LIB indicates that the procedure is in a DLL. The executable code for this procedure is stored in "user.exe" within the DLL. The ( ) indicate an empty parameter list, and the clause AS INTEGER describes the type of value the function returns.

The second statement is similar, except that a SUB procedure does not return a value. The parameters must be immediately followed by an AS clause, as shown by (win AS INTEGER).

Note If a DLL function or subroutine uses an unsupported variable type, you cannot use it in a macro.

See Also

CALL, FUNCTION, SUB, I2W, W2I

DO...LOOP

Description

Repeats statements while a condition is true or until it becomes true. Note that the first form of syntax may never execute statements if expression is/isn't true, but that the second form always executes statements at least once.

Syntax

DO {WHILE | UNTIL} expression
    statements
LOOP

DO
    statements
LOOP {WHILE | UNTIL} expression

Parameters

expression - An expression that is evaluated to determine whether to continue in the loop.

statements - One or more macro statements to be repeated.

Example

```
i% = 5
DO WHILE i% > 0
   i% = i% - 1
LOOP
```

The variable i% starts with a value of 5. The loop continues for as long as i% has a value greater than 0. As soon as i% reaches 0, processing continues at the next statement after the LOOP statement. In this example, the loop executes five times. If the condition were changed to "i% = 0", the loop would never execute because the condition would be true immediately

```
i% = 5
DO UNTIL i% = 0
   i% = i% - 1
LOOP
```

This example is functionally identical to the previous example. The loop executes five times before the condition is true. If the condition were changed to "i% > 0", the loop would never execute.

```
i% = 5
DO
   i% = i% - 1
LOOP UNTIL i% = 0
```

By placing the condition at the end of the loop, the loop will always be executed once before the condition is tested. However, in the following example, the loop will still execute five times.

```
i% = 5
DO
   i% = i% - 1
LOOP WHILE i% > 0
```

Again, the loop will execute five times before the condition is no longer true. If the condition were changed to "i% = 0" the loop would execute only once before the condition is tested.

See Also

FOR...NEXT, WHILE...WEND

END

Description

Ends a SELECT CASE, FUNCTION, SUB, or DIALOG.

Syntax

END {SELECT | FUNCTION | SUB | DIALOG}

Parameters

You must specify the type of subgroup being ended.

Example

```
SELECT CASE i%
   CASE 0
      REM Case statements go here.
END SELECT
FUNCTION UserFunction%( )
   REM Function processing statements go here.
END FUNCTION
SUB Subroutine( )
   REM Subroutine statements go here.
END SUB
BEGIN DIALOG Dialog box1 120, 45, "Sample Dialog box"
   REM Dialog box statements go here.
END DIALOG
```

The macro compiler must match a SELECT CASE, FUNCTION, SUB, or BEGIN DIALOG statement with a corresponding END statement, as shown in the examples above.

See Also

FUNCTION, SELECT CASE, SUB, DIALOG, STOP

FOR...NEXT

Description

Repeats a group of instructions a specified number of times.

Syntax

```
FOR counter% = start TO end [STEP increment]
    statements
NEXT counter%
```

Parameters

counter% - Numeric variable used as counter. Cannot be an array element.

start - Initial value of the counter.

end - Final value of the counter.

increment - Amount the counter is incremented each time through loop (default = 1).

statements - One or more statements to be repeated.

Example

```
FOR i% = 1 TO UBOUND(stringArray$)
   stringArray$(i%) = "string"
NEXT i%
```

All the elements of the array stringArray$ are assigned the value "string".

```
FOR i% = 1 TO 10 STEP 2
   intArray%(i%) = i%
NEXT i%
```

Every other element of the array intArray% (elements 1, 3, 5, 7, and 9) are given the value of i%.

See Also

DO...LOOP, WHILE...WEND

FUNCTION

Description

First line of a user-defined function. If you DECLARE the function with a type-declaration character, then FUNCTION must define it with the same character. Parentheses are required but arguments are not.

Syntax

FUNCTION name{$ | %} ([param{$ | %}] [, param{$ | %}] [,...])

Parameters

name{$ | %} - Name assigned to the function. The type declaration character determines what the function will return (string or integer).

param{$ | %} - Name of one or more variables passed to the function as parameters.

Note Parameters are passed by reference, so if the function changes the parameter's value, the value changes in the calling program.

Example

```
FUNCTION UserFunction$ (str$)
   UserFunction$ = UCASE$(str$)
END FUNCTION
```

This function returns a string that is the uppercase equivalent of the string passed to it.

```
FUNCTION UserFunction2% (i%)
   UserFunction2% = i% ^ 3
END FUNCTION
```

This function returns an integer that is the cube (raised to the 3rd power) of the integer passed to it.

See Also

CALL, DECLARE, SHARED, STATIC, SUB

GOTO

Description

Branches program execution to the line specified by linelabel. A GOTO statement can branch only to another statement at the same level of a program. You cannot use GOTO to enter or exit a SUB or FUNCTION.

Syntax

GOTO {linelabel}

Parameters

linelabel - Label of the line to go to. A label must be followed by a colon and should be no more than 8 characters.

Example

```
REM Statements...
IF i% = 3 THEN GOTO BreakOut
REM Statements execute here if i% is not equal to 3
BreakOut:
REM More statements execute here.
```

If i% is equal to 3, then the statements before the label BreakOut are skipped, and the processing starts with the first statement after the label. The statements after the label will execute even if i% is not equal to 3.

You can use GOTO statements to jump to "clean-up" code before exiting the macro. This code restores specific settings in your macro.

TIP You can also use GOTO to exit from a function or a subroutine by placing a label immediately before the END FUNCTION statement.

See Also

IF...THEN...ELSE...ENDIF

IF...THEN...ELSE...ENDIF

Description

Allows conditional execution based on the evaluation of a condition, i.e., if the condition is true, the program performs the THEN statement; if the condition is false, the program performs the ELSE or ELSEIF statement, if included. IF and THEN are required. If only one THEN statement is needed, and it is included on the same line as the IF condition, then no ENDIF is required. If the THEN statement is more than one line, then ENDIF is required.

Syntax

IF condition1 THEN statement [ELSE statement]

IF condition1 THEN
    statements
[ELSEIF condition2 THEN
    statements]
[ELSE
    statements]
ENDIF

Parameters

condition1, condition2 - a condition to evaluate. If the condition is TRUE, then the statements following the keyword THEN are executed. If the condition is FALSE, then the processing continues at the next ELSEIF statement, if present, or at the statement following the ELSE keyword, if present.

Example

```
IF i% = 0 THEN MESSAGE "The variable is 0." ELSE BEEP
```

If the variable i% has a value of 0, a message appears, otherwise a beep is sounded. Because the entire IF statement is on one line, no ENDIF is needed.

```
IF i% = 0 THEN
   MESSAGE "The variable is 0"
   i% = 1
ELSEIF i% = 1 THEN
   MESSAGE "The variable is less than 2"
ELSEIF i% = 2 THEN
   MESSAGE "The variable is greater than 1"
ELSE
   BEEP
ENDIF
```

You must use the multi-line IF...THEN...ENDIF when there is a second condition to test after the first IF or when there is more than one statement to process as a result of the condition. Repeat the ELSEIF statements for as many conditions as needed. If many conditions must be tested, you might want to use the SELECT CASE statement. Note that in the example, even though i% is assigned a value of 1 in the i% = 0 condition, the ELSEIF i% = 1 condition does not execute. Once a condition is TRUE, processing continues after the ENDIF statement once the statements that meet the condition execute.

See Also

DO...LOOP, SELECT CASE

REM

Description

A non-executed remark, or comment, in a program. You can use an apostrophe (') instead of REM. Everything from REM (or the apostrophe) to the end of the line is a remark. If the first remark line is REM Description: (text), (text) appears in the macro description field of the Customize Keystrokes, Customize Menus, and Customize Toolbox dialog boxes, and in the Status Bar when a macro is highlighted on a menu or Toolbox button.

Syntax

REM This is a comment.

' This is also a comment.

Parameters

comment - any text to describe the macro statements.

Example

```
REM Description: This text will appear in the Status Bar.
REM Filename: filename
REM Created by: Your Name
' This first statement sets up the variables
DIM intArray%(10)
' More processing follows here.
```

SELECT CASE

Description

Compares the value of testexpression to the values in the CASE statements. If none of the values in the CASE statements match testexpression, executes the statements in CASE ELSE.

Syntax

SELECT CASE testexpression
    [CASE {caseexpression | caseexpression TO caseexpression} [,...]
        statements]
    [CASE ELSE
        statements]
END SELECT

Parameters

testexpression - Any numeric or string expression.

caseexpression - An expression that evaluates to the same type as testexpression.

statements -Program instructions conditionally executed; any number of statements on one or more lines.

Example

```
SELECT CASE i%
   CASE -10 TO -5, 5 TO 10
      MESSAGE "Between -10 and -5 or 10 and 5"
   CASE -4 TO -1, 2 TO 4
      MESSAGE "Between -4 and 4 but not 0 or 1"
   CASE 0, 1
      MESSAGE "Zero or one"
   CASE ELSE
      MESSAGE "Greater than 10 or less than -10"
END SELECT
```

If i% is between -10 and -5 or 10 and 5, the message "Between -10 and -5 or 10 and 5" appears. If i% is between -4 and 4, but is not zero or one, the message "Between -4 and 4 but not 0 or 1" appears. If i% is zero or 1, th message "Zero or one" appears. If i% is not any of those numbers, the message "Greater than 10 or less than -10" appears.

See Also

IF...THEN...ELSE...ENDIF

STOP

Description

Stops the macro.

Syntax

STOP

Example

```
IF i% = 3 THEN STOP
```

If the value of i% is 3, then the macro stops immediately. No other statements are executed.

See Also

END

SUB

Description

The required first line of a subprogram definition. Use parentheses if arguments are present.

Syntax

SUB Name ([param{$ | %} [,param{$ | %}...]])

Parameters

name - Any valid name, used to call the subprogram.

param{$ | %} - Name of one or more variables passed to the subprogram. A type-declaration character must indicate the type of argument.

Note Parameters are passed by reference, so if the subroutine changes the parameter's value, the value changes in the calling program.

Example

```
SUB Sub1 (i%, j%, k%)
   k% = i% + j%
END SUB
```

This subroutine assigns the sum of i% and j% to the variable k%. The value of k% will change in the calling routine as well.

See Also

CALL, DECLARE, FUNCTION, SHARED, STATIC

WHILE...WEND

Description

Continuously executes statements as long as the condition is true.

Syntax

WHILE condition
    statements
WEND

Parameters

condition - Any expression that can be called true or false.

statements - One or more macro statements to be executed.

Example

```
i% = 5
WHILE i% > 0
   i% = i% - 1
WEND
```

The loop executes for as long as the condition i% is greater than 0 is true. The loop will execute five times.

See Also

DO...LOOP, FOR...NEXT

String Statements

A string is a series of characters. WSWin uses the ANSI character set. For more information about the ANSI character set, refer to your **Microsoft Windows User's Guide**.

String statements let you define strings, extract parts of strings, compare strings, find and change characters in a string, and convert numbers to strings (and vice versa). Strings must be enclosed in double quotation marks and can be up to 32,766 characters long.

The following string statements are provided by StarBASIC:

- ASC
- CHR$
- DATE$
- INSTR
- LCASE$
- LEFT$
- LEN
- LTRIM$
- MID$
- RIGHT$
- RTRIM$
- SPACE$
- STR$
- STRING$
- TIME$
- UCASE$
- VAL

ASC

Description

Returns the numerical ANSI character value of the first character specified in a string literal, string constant, or string variable. ASC is the opposite of the CHR function, which returns the character when the ANSI value is specified.

Syntax

i% = ASC(string$)

Parameters

string$ - The string to be examined.

Returns

The ANSI value of the first character in the string.

Example

```
i% = ASC("string")
```

This expression will assign the value 115, which is the ANSI value of the letter "s."

See Also

CHR$, STRING$, VAL

CHR$

Description

Returns the ANSI character corresponding to the specified ANSI code value. CHR$ is the opposite of ASC, which returns the ANSI code value when the character is entered.

Syntax

s$ = CHR$ (val%)

Parameters

val% - ASCII code value to be examined.

Returns

The character that occupies the specified position in the ANSI character set.

Example

```
s$ = CHR$(65)
```

Assigns the letter "A" to the variable s$. (Character 65 of the ANSI character set is A.)

Note This function is often used to add special characters to string variables that cannot be entered directly within double quotes. For example, to add double quote marks to a string in a string variable, you use character 34:

```
s$ = chr$(34) + "This will be in double quotes." + chr$(34)
```

You can also use the function to add a carriage return within a string, with character 13:

```
s$ = "String 1" + chr$(13) + "String 2"
MESSAGE s$
```

The result will be the two strings on separate lines, as displayed in the message box.

See Also

ASC, SPACE$, STRING$

DATE$

Description

Returns the current date in the format mm/dd/yy.

Syntax

s$ = DATE$

Returns

The current system date.

Example

```
s$ = DATE$
month% = VAL(LEFT$(s$, 2))
day% = VAL(MID$(s$, 4, 2))
year% = VAL(RIGHT$(s$, 2))
```

The current date is stored in the variable s$. The portions of the date are extracted from the string and converted to integers for the month, day, and year.

See Also

TIME$

INSTR

Description

Returns the position of the first occurrence of string2 within string1.

Syntax

i% = INSTR ([start%,] string1$, string2$)

Parameters

start% - A numeric constant or an integer that specifies the position where the search begins; if unspecified, the search starts at the beginning of string2 (same as start% = 1).

string1$ - String within which the search is made.

string2$ - String searched for.

Returns

The starting position of string2 within string1. If string2 is not found in string1, the function returns 0.

Example

```
pos% = INSTR("Los Angeles", "Ang")
```

Sets pos% to the value 5 because "Ang" occurs at the fifth character in the string "Los Angeles".

See Also

LEFT$, MID$, RIGHT$

LCASE$

Description

Converts a string to lowercase.

Syntax

s$ = LCASE$(source$)

Parameters

source$ - Any string variable, constant, literal, or expression.

Returns

The value of source$, converted to lowercase.

Example

```
lowString$ = LCASE$("ABCDE")
```

Sets lowString$ to "abcde".

See Also

UCASE$

LEFT$

Description

Returns the specified number of characters from the beginning of a string.

Combine LEFT$ with INSTR to extract the portion of a string either up to or including a specified substring.

Syntax

a$ = LEFT$ (source$, num%)

Parameters

source$ - A string variable, constant, literal, or expression.

num% - Number of characters to be returned.

Returns

The first num% characters of source$

Example

```
strleft$ = LEFT$("I want to dance with you", 15)
```

Takes the first 15 characters from "I want to dance with you", which makes the string variable strleft$ equal to "I want to dance".

```
string1$ = "San Francisco, California"
str$ = LEFT$(string1$, INSTR(string1$, ",")-1)
```

Extracts the characters in string1$ up to, but not including, the comma and places the result in the variable str$. The variable str$ now has the value "San Francisco".

Note Subtract 1 from the result of INSTR if you do not want to include the character you are searching for.

See Also

INSTR, MID$, RIGHT$

LEN

Description

Returns the number of characters in a string.

Syntax

i% = LEN (string$)

Parameters

string$ - A string variable, constant, literal, or expression.

Returns

The number of characters in string$.

Example

```
num% = LEN("This is a test")
```

Assigns the length of the string, 14, to the variable num%.

LTRIM$

Description

Removes any leading spaces from a string.

Syntax

a$ = LTRIM$(string$)

Parameters

string$ - A string variable, constant, literal, or expression.

Returns

The value of string$ with all leading spaces removed.

Example

```
str$ = "     Test"
str$ = LTRIM$(str$)
```

Assigns "Test" to the variable str$. All leading spaces that were previously in the variable are removed.

See Also

RTRIM$

MID$

Description

If used as a function, returns a specified number of characters, starting at a specified position in a string.

If used as a statement, assigns a substring to another string, beginning at the specified character.

Syntax

When used as a function, the syntax is:

a$ = MID$ (source$, index% [,count%])

When used as a statement, the syntax is:

MID$ (source$, index% [,count%]) = a$

Parameters

source$ - The original string.

index% - Position of the first character to be copied.

count% - Number of characters to be copied.

Returns

The function returns substring that is count% characters long from source$ starting at index%. If you don't specify count%, the rest of source is returned.

The statement modifies source$, starting at index%, with the characters in a$. If you specify count%, then only count% characters from a$ are copied in source$.

Example

Function example:

```
s$ = MID$("I want to dance with you",11,5)
```

The function extracts five characters from the string, beginning with the eleventh character. The variable s$ then becomes "dance".

Statement example:

```
str1$ = "I want to dance with you"
MID$(str1$, 22, 3) = "him"
```

The statement changes three characters, starting with the 22nd character, to the new string. The str1$ variable now contains "I want to dance with him".

See Also

LEFT$, RIGHT$, INSTR

RIGHT$

Description

Returns the specified number of characters from the end of a string.

Syntax

a$ = RIGHT$ (source$, num%)

Parameters

source$ - A string variable, constant, literal, or expression.

num% - Number of characters to be returned.

Returns

The rightmost num% characters from source$

Example

```
strgrt$ = RIGHT$ ("I don't want to dance",13)
```

Takes the rightmost 13 characters from "I don't want to dance", which makes the string variable strgrt$ equal to "want to dance".

See Also

LEFT$, MID$

RTRIM$

Removes any trailing spaces from a string.

Syntax

a$ = RTRIM$(string$)

Parameters

string$ - A string variable, constant, literal, or expression.

Returns

The value of string$ with all leading spaces removed.

Example

```
str$ = "Test     "
str$ = LTRIM$(str$)
```

Assigns "Test" to the variable str$. All trailing spaces that were previously in the variable are removed.

See Also

LTRIM$

SPACE$

Description

Returns a string consisting of the specified number of spaces (character 32).

Syntax

SPACE$ (num%)

Parameters

num% - Number of spaces to be returned.

Returns

A string consisting of num% spaces.

Example

```
str$ = SPACE$(4) + "Test" + SPACE$(4)
```

Makes the string variable str$ equal to "   Test   ".

See Also

LTRIM$, RTRIM$

STR$

Description

Returns a string representation of an integer. This allows for manipulation of the number as a string of characters, not as integers.

Syntax

STR$ (num%)

Parameters

num% - Any numeric variable, constant, literal, or expression.

Returns

The string representation of the integer num%.

Example

```
anumstr$ =  STR$(72)
```

Assigns "72" to the variable anumstr$.

See Also

VAL

STRING$

Description

STRING$ (num%, string$)

Syntax

Returns the first character of string$ a specified num% of times.

Parameters

num% - Number of times the character is to be returned.

string$ - String to be examined.

Returns

A string containing the first character of string$, repeated num% times.

Example

```
a$ = STRING$(10, "x")
Assigns "xxxxxxxxxx" to the variable a$.
```

See Also

CHR$, SPACE$

TIME$

Description

Returns string representation of current time in 24-hour mode (hh:mm:ss).

Syntax

s$ = TIME$

Returns

The current system time.

Example

```
s$ = TIME$
hour% = VAL(LEFT$(s$, 2))
min% = VAL(MID$(s$, 4, 2))
sec% = VAL(RIGHT$(s$, 2))
```

The current time is stored in the variable s$. The portions of the time are extracted from the string and converted to integers for the hours, minutes, and seconds.

See Also

DATE$

UCASE$

Description

Converts a string to uppercase.

Syntax

s$ = UCASE$(source$)

Parameters

source$ - Any string variable, constant, literal, or expression.

Returns

The value of source$, converted to uppercase.

Example

```
upperString$ = UCASE$("abcde")
```

Sets upperString$ to "ABCDE".

See Also

LCASE$

VAL

Description

Converts a string in its numeric equivalent. This function is the opposite of the STR$ statement.

Syntax

i% = VAL (chr$)

Parameters

chr$ - The string to be converted.

Returns

The integer represented by the string

Example

```
i% = VAL("72")
```

Assigns 72 to the variable i%. Because edit boxes in macro dialog boxes can only obtain strings, you can use this function to convert strings entered in dialog boxes to integers.

See Also

STR$

DOS and File Statements

The DOS and file statements provide the essential connection between your macro and the operating system. These statements allow you to manipulate external files and create and remove directories.

You can have up to ten files open at one time. These files are assigned numbers from 1 to 10 when they are opened. Any time you refer to a file after it is opened, you use the number of the file rather than the filename.

The following DOS and file statements are available:

- ACCESS
- CHDIR
- CLOSE
- EOF
- FILEATTR
- FILEPOS
- FREEFILE
- INPUT$
- INPUT
- KILL
- LINE INPUT
- LOF
- MKDIR
- OPEN
- PRINT
- RMDIR
- SEEK
- WRITE

ACCESS

Description

Returns the file-access permission of the specified file.

Syntax

retval% = ACCESS(filename$, mode%)

Parameters

filename$ - The name of the file to access. You can include the drive and directory.

mode% - the access mode to use. Specify 0 to check for the existence of the file, 2 to check for write permission, 4 to check for read permission, or 6 to check for read and write permission.

Returns

The function returns TRUE if the specified file is available in the given mode. If the file does not exist or is not available in the specified mode, the function returns FALSE.

Example

```
i% = ACCESS("letter.wsd", 0)
```

If LETTER.WSD exists, i% is set to TRUE. If it does not exist, i% is set to FALSE.

```
i% = ACCESS("memo.wsd", 2)
```

If MEMO.WSD is read-only, i% is set to FALSE. If it has write access, i% is set to TRUE.

See Also

OPEN, FILEATTR

CHDIR

Description

Changes current drive and directory. This statement changes the current directory only for the DOS and File Statements listed below. To change the current directory for WSWin file operations (FileOpen, FileSave, etc.), use the SetCurrentDir statement.

Syntax

CHDIR dirName$

Parameters

dirName$ - Name of the desired directory.

Example

```
CHDIR "\WORK"
```

Changes the current directory to "\WORK".

```
CHDIR dir1$
```

Changes the current directory to the value in the variable dir1$.

See Also

MKDIR, RMDIR, SetCurrentDir

CLOSE

Description

Closes all open files, if no file is specified, or closes a specific numbered file.

Syntax

CLOSE [[#]num%[,...]]

Parameters

num% - File number of the file to be closed; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Example

```
CLOSE
```

Closes all open files

```
CLOSE #1
```

Closes the file opened as 1.

```
CLOSE 1
```

Closes the file opened as 1.

```
CLOSE 1, 3, 5
```

Closes the files opened as 1, 3, and 5.

See Also

OPEN

EOF

Description

Returns TRUE if the file pointer is at the end of the file. Returns FALSE if the file contains data beyond the pointer. Used to determine whether to continue processing a file.

Syntax

test% = EOF ([#]num%)

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Returns

TRUE if the pointer is at the end of file num%.

Example

```
IF EOF(1) THEN CLOSE 1
```

If the pointer is at the end of the file, then close the file.

See Also

LOF, SEEK

FILEATTR

Description

Returns either the file mode or the DOS file handle (the internal number DOS uses to identify a file) of an open file. Used, for example, to determine whether a file is "read only."

Note This is the only StarBASIC file function that doesn't have an optional # sign in front of the file number.

Syntax

retval% = FILEATTR(num%, attribute%)

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive.

attribute% - 1 returns information about the mode in which the file was opened (see below); 2 returns the file's DOS file handle.

Returns

If attribute% is 1, the function returns one of the following:

1 if file num% was opened for input

2 if file num% was opened for output

8 if file num% was opened for append

If attribute% is 2, the function returns the DOS file handle of file num%

Example

```
i% = FILEATTR(1, 1)
```

If file 1 was opened for input, then i% is set to 1. If file 1 was opened for output, then i% is set to 2. If file 1 was opened for append, then i% is set to 8.

```
i% = FILEATTR(1, 2)
```

Returns the DOS file handle for file 1.

See Also

ACCESS,, OPEN

FILEPOS

Description

Returns the current file position for the specified file.

Syntax

retval% = FILEPOS ([#]num%)

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Returns

The current file position for file num%.

Example

```
pos% = FILEPOS(1)
```

Sets pos% equal to the current position of the file pointer for file 1.

See Also

SEEK

FREEFILE

Description

Returns the lowest file number not associated with an open file.

Syntax

retval% = FREEFILE

Returns

The lowest available file number not currently in use.

Example

```
OPEN "temp.out" FOR OUTPUT AS 1
OPEN "temp2.out" FOR OUTPUT AS 5
i% = FREEFILE
```

The lowest available file number in this example is 2 because 1 is already being used, so i% is set to 2.

See Also

OPEN

INPUT$

Description

Starting at the pointer, reads a number of bytes from a file.

Syntax

string$ = INPUT$(bytes%, [#]num%)

Parameters

bytes% - The number of bytes to be read.

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Returns

A string containing the specified number of bytes from file num%.

Example

```
string$ = INPUT$(50, 1)
```

Reads 50 characters from file 1 and assigns them to the variable string$.

See Also

INPUT, LINE INPUT, SEEK, FILEPOS

INPUT

Description

Reads from a file to a list of variables. Values in the file are separated by commas. Character strings that include commas must be enclosed in quotation marks. The quotation marks do not appear in variable.

Syntax

INPUT[#]num%, var1{$ | %} [, var2{$ | %}, ...]

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

var1{$ | %}, var2{$ | %} - the variables that receive the fields as they are read from the file.

Example

```
INPUT 1, title$, number%
```

Reads a string and an integer from file 1. Assigns the values to the string variable title$ and the integer variable number%. The contents of file 1 could be in either of two formats:

```
"To Kill a Mockingbird", 50
"Huckleberry Finn", 101
"Hamlet", 220
```

or

```
"To Kill a Mockingbird", 50, "Huckleberry Finn", 101, "Hamlet", 220
```

See Also

INPUT$, LINE INPUT

KILL

Description

Deletes a file. This statement is the same as the DEL or ERASE command in DOS.

Syntax

KILL fileName$

Parameters

fileName$ - Name of the file to be deleted.

Example

```
KILL "temp.out"
```

Deletes the file "temp.out" in the current directory.

See Also

RMDIR

LINE INPUT

Description

Reads the next line from a file in a string.

Syntax

LINE INPUT [#]num%, string$

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

string$ - The string variable to hold the line from the file.

Example

```
LINE INPUT 1, string$
```

Reads the next line from file 1 and places the string in the variable string$.

See Also

INPUT, INPUT$

LOF

Description

LOF (Length of File) returns the number of bytes in a file.

Syntax

retval% = LOF([#]num%)

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

Returns

The number of bytes in file num%

Example

```
bytes% = LOF(1)
```

Sets bytes% to the number of bytes in file 1.

See Also

OPEN

MKDIR

Description

Creates a new subdirectory.

Syntax

MKDIR dirName$

Parameters

dirName$ - Name of the subdirectory to be created.

Example

```
MKDIR "work"
```

Creates the directory **work** as a subdirectory of the current directory.

See Also

CHDIR, RMDIR

OPEN

Description

Opens a file using a specified number. Specifying INPUT opens the sequential file for content access. Specifying OUTPUT creates a sequential file for output. Specifying APPEND opens a sequential file for adding data to the end of the file. The FOR clause is required.

Syntax

OPEN fileName$ FOR {INPUT | OUTPUT | APPEND} AS [#]num%

Parameters

fileName$ - Name of the file to open.

num% - Number (1 through 10) associated with the file; used to access the file.

Example

```
OPEN "\temp\workfile.tmp" FOR OUTPUT AS 1
OPEN string$ FOR INPUT AS 2
OPEN "oldfile" FOR APPEND AS 3
```

Opens the file \temp\workfile.tmp for output, the filename specified by string$ for input, and oldfile for append.

See Also

ACCESS, CLOSE, EOF, FILEATTR, FILEPOS, FREEFILE, INPUT$, INPUT, LINE INPUT, LOF, PRINT, SEEK, WRITE

PRINT

Description

Prints expression(s) to a file. Character strings are not enclosed in quotes when written out. Numeric expressions have either a leading space or a negative sign in addition to a trailing blank space. Each expression follows immediately after the previous one. If expressions are separated by a comma, objects are put in the file separated by spaces that align them to a tab stop. If expressions are separated by a semicolon, objects are put in the file with no spaces between them.

Syntax

PRINT [#]num%, expression1 [{, | ;} expression2 {, | ;} ...]

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

expression1, expression2 - String or numeric expressions to be written to the file.

Example

```
PRINT 1, "HEADING:  "; string1$; ","; int%
```

Outputs "HEADING: " followed by the value of string1$, a comma, and the value in int%. No spaces are added between the four items, except for those contained in the literal string.

See Also

INPUT, WRITE

RMDIR

Description

Removes an empty subdirectory.

Syntax

RMDIR dirName$

Parameters

dirName$ - The name of the subdirectory to be removed.

Example

```
RMDIR "\TEMP\WORK"
```

Removes the WORK directory from the \TEMP directory.

See Also

CHDIR, KILL, MKDIR

SEEK

Description

To prepare for subsequent input, moves the file pointer to a specific byte position in a file.

Syntax

SEEK [#]num%, position%

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

position% - The byte position to SEEK (1 is 1st byte).

Example

```
SEEK 1, 100
```

Moves the file pointer to the 100th byte in file 1.

See Also

OPEN

WRITE

Description

Writes expressions to a file. Character strings are enclosed in quotes when written out. Each expression is separated by a comma.

Syntax

WRITE [#]num%, expression1 [, expression2, ...]

Parameters

num% - File number of the file to be examined; a constant or an integer variable with the value 1-10, inclusive. The # sign is optional.

expression1, expression2 - The expressions to write to the file.

Example

```
string$ = "some text"
int% = 55
WRITE 1, string$, "Literal Text", int%
```

Writes the following to file 1: "some text", "Literal Text", 55.

See Also

PRINT

User Interface Statements

You can use user interface statements to define and display several kinds of interactive dialog boxes or to sound a tone.

Note The statements INPUTBOX$, MESSAGEBOX, GETFILEBOX$, and any user-defined dialog boxes that include a Cancel button involve a system variable called CANCEL. When the macro starts, this variable is set to zero. If the Cancel button is chosen, the variable is set to TRUE. The value of the CANCEL variable remains unchanged between statement calls, unless the macro changes it. A typical use of this variable would be

```
IF CANCEL THEN STOP
```

StarBASIC provides the following user interface statements:

- ABOUT
- BEEP
- GETFILEBOX$
- INPUTBOX$
- MESSAGE
- MESSAGEBOX
- BEGIN DIALOG
- CANCELBUTTON
- CHECKBOX
- COMBOBOX
- DIALOG
- DIRECTORYBOX
- DROPBOX
- END DIALOG
- GROUPBOX
- LISTBOX
- OKBUTTON
- OPTIONBUTTON
- OPTIONGROUP
- PUSHBUTTON
- TEXT
- TEXTBOX.

ABOUT

Description

Displays the About dialog box for the WSWin macro interpreter.

Syntax

ABOUT

BEEP

Description

Sounds a tone.

Syntax

BEEP

GetFileBox$

Description

Displays a dialog box which allows the user to select a file from the file system. Returns the selected filename or an empty string if the user chooses Cancel.

Syntax

fileName$ = GetFileBox$ (filespec$, title$)

Parameters

filespec$ - Specifies the files to display in the box. This can include a default directory and wild-card characters. You can sepate multiple extensions with a semicolon (;).

title$ - The title to display in the dialog box.

Returns

The filename chosen by the user.

Example

```
CHDIR "c:\wswin\docs"
filename$ = GetFileBox$("*.*", "Choose a File")
```

The standard file dialog box appears with all the files in the current directory displayed in the file list.

```
filename$ = GetFileBox("c:\wswin\docs\*.wsd;*.wst", "Choose a Document or
   Template")
```

The file dialog box appears with all files with either a WSD or WST extension in the directory C:\WSWIN\DOCS displayed in the list.

See Also

CHDIR

INPUTBOX$

Description

Displays a simple dialog box where the user can enter a string. The dialog box has OK and Cancel buttons.

Syntax

string$ = INPUTBOX$(prompt$)

Parameters

prompt$ - String variable, constant, expression, or literal that appears in the dialog box above the edit box.

Returns

The string entered in the edit box by the user. If the Cancel button is chosen, an empty string is returned.

Example

```
string$ = INPUTBOX$("Please type in a string")
```

MESSAGE

Description

Displays a box containing the user-specified message and an OK button.

Syntax

Message string$

Parameters

string$ - The string to display in the message box.

Example

```
message  "What a nice day."
```

See Also

MESSAGEBOX

MESSAGEBOX

Description

Displays a message box with a user-specified message and user-specified buttons and icon.

Syntax

retval% = MESSAGEBOX(prompt$, title$, option%)

Parameters

prompt$ - The string to display in the box.

title$ - The string to display in the message box caption.

option% - A number representing the type of buttons to include in the box and an icon (if any) to appear beside the message. The option% value can be the sum of the following values:

Button Type

- 0 OK only
- 1 OK/Cancel
- 2 Abort/Retry/Ignore
- 3 Yes/No/Cancel
- 4 Yes/No
- 5 Retry/Cancel

Icon Type

- 0 No icon
- 16 Stop
- 32 Question
- 48 Exclamation
- 64 Information (lowercase i)

Returns

The number of the button pressed by the user (1 = OK, 2 = Cancel, 3 = Abort, 4 = Retry, 5 = Ignore, 6 = Yes, 7 = No).

Example

```
retval% = MESSAGEBOX("An error has occurred", "Error", 2 + 16)
```

Displays the message "An error has occurred" with Abort, Retry, and Ignore buttons and a stop icon.

See Also

MESSAGE

BEGIN DIALOG

The remaining user interface statements refer to user-defined dialog boxes. User-defined dialog boxes always start with a BEGIN DIALOG statement and end with an END DIALOG statement.

Description

Defines a user-defined dialog box as determined by statements. Use the DIALOG command to display the dialog box and wait for user input.

Syntax

BEGIN DIALOG dialog boxName [originX%, originY%,] width%, height%, text$
    (dialog box statements)
END DIALOG

Parameters

dialog boxName - Name assigned to the dialog box sequence.

originX% - Location of box, horizontally, in current dialog base width units, from the upper-left corner of the screen.

originY% - Location of box, vertically, in current dialog base height units, from the upper-left corner of the screen.

(If you omit the origin points, the box will be centered on the screen.)

width% - Width of the dialog box from the origin, in current dialog base width units.

height% - Height of the dialog box from the origin, in current dialog base height units.

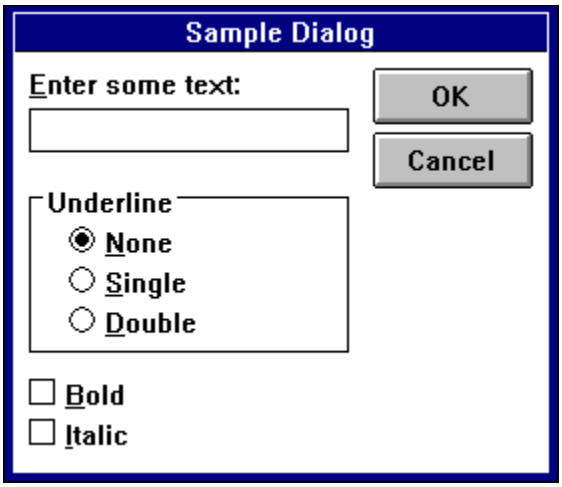
text$ - Text to appear as a caption for the dialog box.

statements - The combination of statements that define a dialog box. Can include OKBUTTON, CANCELBUTTON, TEXT, LISTBOX, COMBOBOX, DROPBOX, DIRECTORYBOX, CHECKBOX, GROUPBOX, and OPTIONGROUP.

Example

```
BEGIN DIALOG Dialog box1 134, 105, "Sample Dialog box"
   TEXT           4, 4, 80, 8, "&Enter some text:"
   TEXTBOX        4, 14, 80, 11, text$
   GROUPBOX       4, 32, 80, 43, "Underline"
   OPTIONGROUP    underline%
   OPTIONBUTTON   14, 42, 60, 10, "&None"
   OPTIONBUTTON   14, 52, 60, 10, "&Single"
   OPTIONBUTTON   14, 62, 60, 10, "&Double"
   CHECKBOX       4, 80, 60, 10, "&Bold", bold%
   CHECKBOX       4, 90, 60, 10, "&Italic", ital%
   OKBUTTON       90, 4, 40, 14
   CANCELBUTTON   90, 20, 40, 14
END DIALOG
ret% = DIALOG(Dialog box1)
```

Displays the following dialog box:



See Also

DIALOG, OKBUTTON, CANCELBUTTON, PUSHBUTTON, TEXT, TEXTBOX, LISTBOX, COMBOBOX, DROPBOX, DIRECTORYBOX, CHECKBOX, GROUPBOX, OPTIONBUTTON, OPTIONGROUP, END DIALOG

CANCELBUTTON

Description

Displays a Cancel button in the dialog box. If the user chooses this button, the DIALOG function returns 2.

Syntax

CANCELBUTTON originX%, originY%, width%, height%

Parameters

originX% - Location of the button horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the button vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the button in current dialog base width units.

height% - Height of the button in current dialog base height units.

Example

See BEGIN DIALOG.

See Also

OKBUTTON, PUSHBUTTON

CHECKBOX

Description

Displays a box the user can check. Returns TRUE if checked, FALSE if not checked.

Syntax

CHECKBOX originX%, originY%, width%, height%, text$, chk%

Parameters

originX% - Location of box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of box in current dialog base width units.

height% - Height of box in current dialog base height units.

text$ - Text to appear next to the check box, such as "&Italic", "&Bold", "&Underline". The ampersand (&) specifies the character to underline.

chk% - The variable containing the return value from the check box.

Example

See BEGIN DIALOG.

See Also

OPTIONBUTTON

COMBOBOX

Description

Returns the text of the selected combo box item.

Syntax

COMBOBOX originX%, originY%, width%, height%, arr$, combo$

Parameters

originX% - Location of the combo box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the combo box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the combo box in current dialog base width units.

height% - Height of the combo box in current dialog base height units.

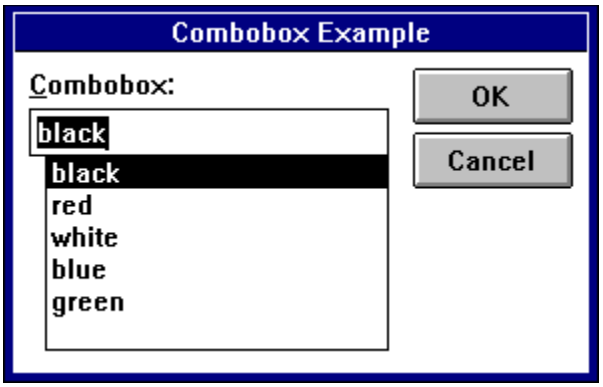
arr$ - An array of items to be displayed in the combo box. The items must be defined elsewhere in the program and specified using the DIM command. The list stops at the first empty array element.

combo$ - The string containing the returned text from the selected combo box. it can match an element from the array or not, if the user types in a different value. If initialized to some string, the first matching element is selected in the combobox.

Example

```
DIM arr$ (5)
arr$(1) = "black"
arr$(2) = "red"
arr$(3) = "white"
arr$(4) = "blue"
arr$(5) = "green"
combo$ = arr$(1)
BEGIN DIALOG combodlg 144, 80, "Combobox Example"
   TEXT           4, 4, 90, 8, "&Combobox:"
   COMBOBOX       4, 14, 90, 62, arr$, combo$
   OKBUTTON       100, 4, 40, 14
   CANCELBUTTON   100, 20, 40, 14
END DIALOG
ret% = DIALOG(combodlg)
' If Cancel is selected, stop the macro
IF ret% = 2 THEN STOP
FOR i% = 1 TO 6
   ' If we are within the bounds of the array and
   ' find a match, then display a message.
   IF (i% <= 5) AND (combo$ = arr$(i%)) THEN
      MESSAGE "You chose " + combo$
      GOTO ENDFOR
   ENDIF
   REM If we get here, then we didn't get a match.
   IF i% = 6 THEN
      MESSAGE "You typed in your own color"
   ENDIF
ENDFOR:
NEXT i%
```

Displays the dialog box until user selects an item from arr$ and selects OK. The text is returned in the variable combo$.



See Also

DIRECTORYBOX, DROPBOX, LISTBOX

DIALOG

Description

Displays a user-defined dialog box, using the dialog box definition established earlier in the macro.

Syntax

ret% = DIALOG(userDialog box)

Parameters

userDialog box - Name assigned to the dialog box sequence.

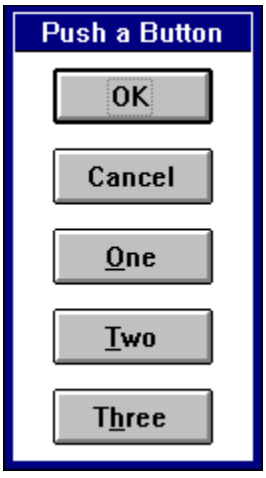
Returns

The number associated with the pushbutton pressed by the user to close the macro. The OKBUTTON always returns 1 and the CANCELBUTTON always returns 2. A PUSHBUTTON control returns the number assigned to the pushbutton.

Example

```
BEGIN DIALOG Dialog box1 60, 102, "Push a Button"
   OKBUTTON       10, 4, 40, 14
   CANCELBUTTON   10, 24, 40, 14
   PUSHBUTTON     10, 44, 40, 14, "&One", 3
   PUSHBUTTON     10, 64, 40, 14, "&Two", 4
   PUSHBUTTON     10, 84, 40, 14, "T&hree", 5
END DIALOG
ret% = DIALOG(Dialog box1)
' Display message depending on which button was pushed
SELECT CASE ret%
   CASE 1
      MESSAGE "You pushed OK"
   CASE 2
      MESSAGE "You pushed Cancel"
   CASE 3
      MESSAGE "You pushed One"
   CASE 4
      MESSAGE "You pushed Two"
   CASE 5
      MESSAGE "You pushed Three"
END SELECT
```

Displays a dialog box of pushbuttons. A message is displayed based on which pushbutton the user chooses.



See Also
BEGIN DIALOG, END DIALOG

DIRECTORYBOX

Description

Displays a combo box that uses the automatic directory listing capability of Windows.

Syntax

DIRECTORYBOX originX%, originY%, width%, height%, dirs%, file$

Parameters

originX% - Location of the box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the box in current dialog base width units.

height% - Height of the box in current dialog base height units.

dirs% - Type of files to list; can be any of the following values (or a sum of any of these):

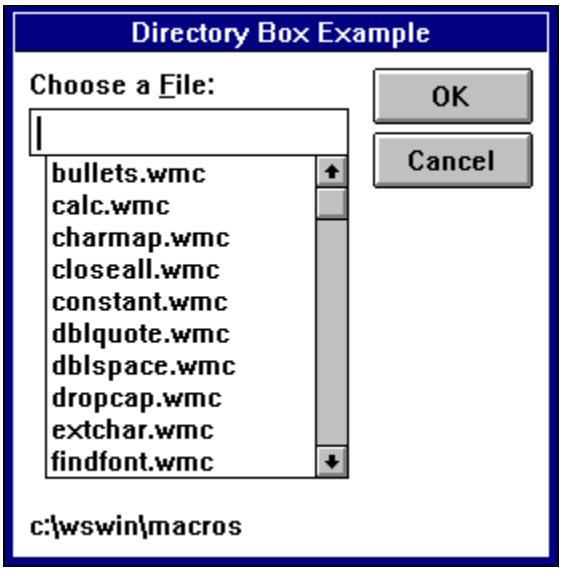
0	Normal files
1	Include read-only files
2	Include hidden files
4	Include system files
16	Include subdirectories
32	Include files with the archive-bit set
16834	Include drive letters
32768	List only the files that match the type specification

file$ - The variable containing the returned string variable of an item selected in the directory box; may initially contain wild-card characters and characters to limit the file search when the dialog box is initialized (e.g., "*.wsd").

Example

```
' Show all macro files, directories, and drives
file$ = "*.wmc"
dirs% = 1 + 2 + 4 + 16 + 32 + 16384
currentDir$ = GetCurrentDir$
BEGIN DIALOG dirdlg 134, 126, "Directory Box Example"
   TEXT           4, 4, 80, 8, "Choose a &File:"
   DIRECTORYBOX   4, 14, 80, 96, dirs%, file$
   TEXT           4, 114, 80, 8, currentDir$
   OKBUTTON       90, 4, 40, 14
   CANCELBUTTON   90, 20, 40, 14
END DIALOG
ret% = DIALOG(dirdlg)
```

Displays a dialog box listing files, directories, and drives in the current directory, as shown below.



See Also

COMBOBOX, DROPBOX, LISTBOX

DROPBOX

Description

Displays a "drop-down" combo box.

Syntax

DROPBOX originX%, originY%, width%, height%, arr$, drop$

Parameters

originX% - Location of the box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the box in current dialog base width units.

height% - Height of the box in current dialog base height units.

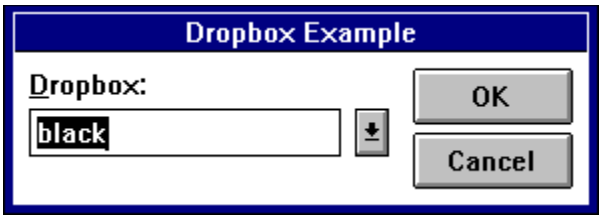
arr$ - Name of the array of items to be displayed in the drop-down box. These items must be defined elsewhere in the program, and specified using the DIM command. The list stops at the first empty array element.

drop$ - Text returned from user's selection in the drop-down box. To choose a default entry for the dropbox, initialize drop$ to one of the values in arr$.

Example

```
DIM arr$ (5)
arr$(1) = "black"
arr$(2) = "red"
arr$(3) = "white"
arr$(4) = "blue"
arr$(5) = "green"
combo$ = arr$(1)
BEGIN DIALOG dropdlg 144, 38, "Dropbox Example"
   TEXT           4, 4, 90, 8, "&Dropbox:"
   DROPBOX        4, 14, 90, 62, arr$, combo$
   OKBUTTON       100, 4, 40, 14
   CANCELBUTTON   100, 20, 40, 14
END DIALOG
ret% = DIALOG(dropdlg)
' If Cancel is selected, stop the macro
IF ret% = 2 THEN STOP
FOR i% = 1 TO 6
   ' If we are within the bounds of the array and
   ' find a match, then display a message.
   IF (i% <= 5) AND (combo$ = arr$(i%)) THEN
      MESSAGE "You chose " + combo$
      GOTO ENDFOR
   ENDIF
   REM If we get here, then we didn't get a match.
   IF i% = 6 THEN
      MESSAGE "You typed in your own color"
   ENDIF
ENDFOR:
NEXT i%
```

Displays a dialog box with a drop-down combobox, as shown below. The combobox is initialized to the first element of an array of strings.



See Also

COMBOBOX, DIRECTORYBOX, LISTBOX

END DIALOG

Description

Ends the definition of a user-defined dialog box. Use the DIALOG function to display the box.

Syntax

```
BEGIN DIALOG dialog boxName [origin X%, origin Y%,] width%, height%, [text$]
    (dialog box statements)
END DIALOG
```

Example

See BEGIN DIALOG.

See Also

DIALOG

GROUPBOX

Description

Draws a labeled box. Normally used to enclose a group of related OPTIONBUTTON or CHECKBOX statements.

Syntax

GROUPBOX originX%, originY%, width%, height%, text$

Parameters

originX% - Location of the box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the box in current dialog base width units.

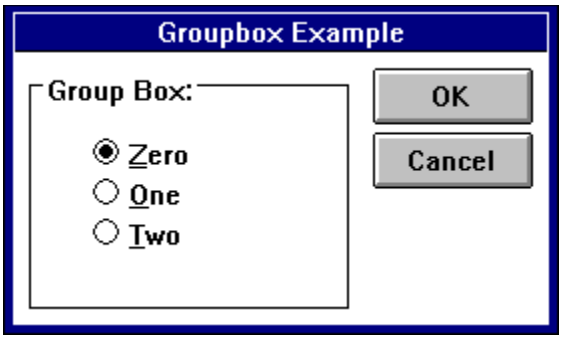
height% - Height of the box in current dialog base height units.

text$ - Label of group box.

Example

```
BEGIN DIALOG grpdlg 134, 68, "Groupbox Example"
   GROUPBOX       4, 4, 80, 60, "Group Box:"
   OPTIONGROUP    ogroup%
   OPTIONBUTTON   20, 20, 30, 10, "&Zero"
   OPTIONBUTTON   20, 30, 30, 10, "&One"
   OPTIONBUTTON   20, 40, 30, 10, "&Two"
   OKBUTTON       90, 4, 40, 14
   CANCELBUTTON   90, 20, 40, 14
END DIALOG
ret% = DIALOG(grpdlg)
' If ret% is 2, then Cancel button was chosen
IF ret% = 2 THEN STOP
SELECT CASE ogroup%
   CASE 0
      ' The Zero option button was selected
      MESSAGE "You chose Zero"
   CASE 1
      ' The One option button was selected
      MESSAGE "You chose One"
   CASE 2
      ' The Two option button was selected
      MESSAGE "You chose Two"
END SELECT
```

Displays the dialog box shown below until the user selects one of the options buttons and OK. The response value is then returned in the variable ogroup%.



See Also

CHECKBOX, OPTIONBUTTON, OPTIONGROUP

LISTBOX

Description

Returns the array index of the selected list box item.

Syntax

LISTBOX originX%, originY%, width%, height%, arr$, index%

Parameters

originX% - Location of the list box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the list box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the list box in current dialog base width units.

height% - Height of the list box in current dialog base height units.

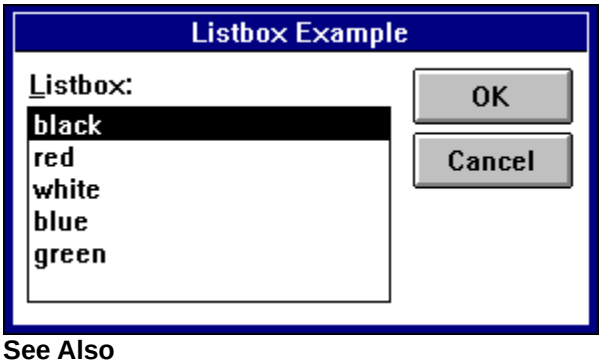
arr$ - Name of the array of items to be displayed in the list box. The items must be defined elsewhere in the program and specified using the DIM command. The list stops at the first empty array element.

index% - The variable containing the return array index of the selected list item. If the variable is initialized to a value, that element of the array will be selected in the list.

Example

```
DIM arr$ (5)
arr$(1) = "black"
arr$(2) = "red"
arr$(3) = "white"
arr$(4) = "blue"
arr$(5) = "green"
index% = 1
BEGIN DIALOG combodlg 144, 68, "List Example"
   TEXT           4, 4, 90, 8, "&List:"
   LISTBOX        4, 14, 90, 50, arr$, index%
   OKBUTTON       100, 4, 40, 14
   CANCELBUTTON   100, 20, 40, 14
END DIALOG
ret% = DIALOG(combodlg)
' If Cancel is selected, stop the macro
IF ret% = 2 THEN STOP
MESSAGE "You chose " + arr$(index%)
```

Displays the dialog box shown below and waits for the user to select an element in the list box and then choose OK.



See Also

COMBOBOX, DIRECTORYBOX, DROPBOX

OKBUTTON

Description

Displays an OK button within the dialog box.

Syntax

OKBUTTON origin X%, origin Y%, width%, height%

Parameters

originX% - Location of the button horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the button vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the button in current dialog base width units.

height% - Height of the button in current dialog base height units.

Example

See BEGIN DIALOG.

See Also

CANCELBUTTON, PUSHBUTTON

OPTIONGROUP

Description

Marks the beginning of a series of OPTIONBUTTONs. You can select only one OPTIONBUTTON in a group. Its value (0 through n) is returned in the OPTIONGROUP variable. An OPTIONGROUP remains in effect and contains all OPTIONBUTTONs until the next OPTIONGROUP statement occurs.

Syntax

OPTIONGROUP ogroup%

Parameters

ogroup% - The number of the option button selected by the user when the dialog box is closed, beginning with 0.

Example

See GROUPBOX.

See Also

OPTIONBUTTON

OPTIONBUTTON

Description

Displays an OPTIONBUTTON within an OPTIONGROUP. Only OPTIONBUTTON in a group can be selected. Its value (0 through n) is returned in the OPTIONGROUP variable.

Syntax

OPTIONBUTTON originX%, originY%, width%, height%, text$

Parameters

originX% - Location of the button horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the button vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the button in current dialog base width units.

height% - Height of the button in current dialog base height units.

text$ - Text to display on the button.

Example

See GROUPBOX.

See Also

OPTIONGROUP

PUSHBUTTON

Description

Displays a pushbutton in the dialog box.

Syntax

PUSHBUTTON origin X%, origin Y%, width%, height%, text$, return%

Parameters

originX% - Location of the button horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the button vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the button in current dialog base width units.

height% - Height of the button in current dialog base height units.

text$ - The text to display on the button.

return% - The numeric value to return in the DIALOG function if the pushbutton is chosen.

Example

See BEGIN DIALOG.

See Also

CANCELBUTTON, OKBUTTON

TEXT

Description

Displays text at a designated location within the dialog box. You can use this command to label boxes, such as a list box.

Syntax

TEXT origin X%, origin Y%, width%, height%, text$

Parameters

originX% - Location of the text box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the text box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the text box.

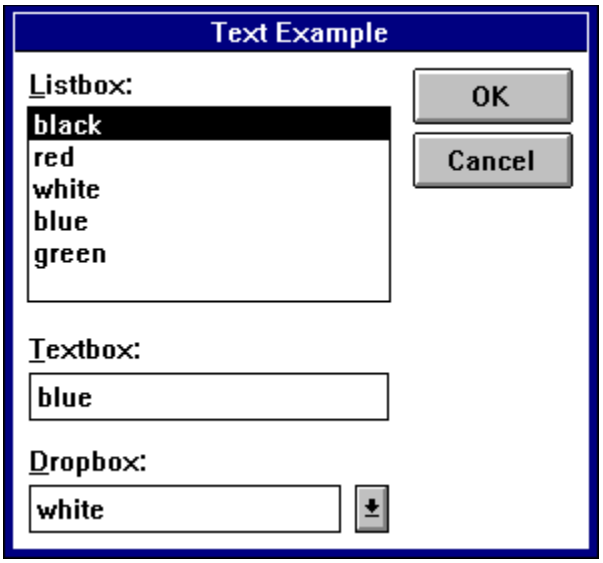
height% - Height of the text box.

text$ - Text to be displayed at the desired location. An "&" in the text string underlines the next letter of the string to be used as the Alt-key mnemonic. When the Alt-key combination is pressed, the dialog box stops at the next control in the dialog box description sequence, based on the order of the controls in the description.

Example

```
DIM arr$ (5)
arr$(1) = "black"
arr$(2) = "red"
arr$(3) = "white"
arr$(4) = "blue"
arr$(5) = "green"
' Carriage return for message
CR$ = CHR$(13)
' Initialize dialog box variables
index% = 1
combo$ = arr$(3)
text$ = arr$(4)
BEGIN DIALOG textdlg 144, 124, "Text Example"
   TEXT           4, 4, 90, 8, "&List:"
   LISTBOX        4, 14, 90, 50, arr$, index%
   TEXT           4, 70, 40, 8, "&Textbox:"
   TEXTBOX        4, 80, 90, 12, text$
   TEXT           4, 98, 90, 8, "&Dropbox:"
   DROPBOX        4, 108, 90, 50, arr$, combo$
   OKBUTTON       100, 4, 40, 14
   CANCELBUTTON   100, 20, 40, 14
END DIALOG
ret% = DIALOG(textdlg)
' If Cancel is selected, stop the macro
IF ret% = 2 THEN STOP
MESSAGE "You chose:" +CR$+ arr$(index%) +CR$+ text$ +CR$+ combo$
```

Displays the dialog box shown below. Note that the mnemonic for each TEXT control will send the dialog box focus to the next control in the dialog box, as indicated by the string for each TEXT control.



See Also

COMBOBOX, DIRECTORYBOX, DROPBOX, LISTBOX, TEXTBOX

TEXTBOX

Description

Displays a text box for entering text.

Syntax

TEXTBOX originX%, originY%, width%, height%, text$

Parameters

originX% - Location of the text box horizontally, in current dialog base width units, from the upper-left corner of the dialog box.

originY% - Location of the text box vertically, in current dialog base height units, from the upper-left corner of the dialog box.

width% - Width of the text box in current dialog base width units.

height% - Height of the text box in current dialog base height units.

text$ - String that returns the text the user enters in the text box; may be initialized.

Example

See TEXT.

File Statements

The File Statements section describes the available commands on the WSWin File menu. The File Statements include

- FileNew
- FileNewDefaults
- FileOpen
- FileClose
- FileImport
- FileSave
- FileSaveAs
- FileSaveAsCopy
- FileExport
- FileRevertToSaved
- FilePageSetup
- FilePrintSetup
- FilePrint
- FilePrintDefaults
- FileExit

FileNew

Description

Creates a new document based on a specified template.

Syntax

FileNew templateName$

Parameters

templateName$ - The name of the template file to use for the new document; can include a full path. If no template is specified, the New dialog box displays.

Example

```
section$ = "Preferences"
key$ = "TemplateDirectory"
file$ = "wsw.ini"
' Get default template directory from the INI file
GetPrivateProfileString(section$, key$, "", tempDir$, file$)
IF tempDir$ <> "" THEN tempDir$ = tempDir$ + "\"
FileNew tempDir$ + "memo.wst"
```

Obtains the default template directory from the WSW.INI file and then creates a new document based on the template MEMO.WST in that directory.

See Also

FileNewDefaults, FileOpen

FileNewDefaults

Description

Creates a new document based on the default template.

Syntax

FileNewDefaults

Example

```
' Print selected text
EditCopy
FileNewDefaults
EditPaste
FilePrintDefaults
FileClose 0, 1
```

This macro copies selected text to the Clipboard, creates a new temporary document, pastes the text to the document, prints, and closes the document.

See Also

FileNew, FileOpen

FileOpen

Description

Opens the specified document.

Syntax

FileOpen fileName$

Parameters

fileName$ - The name of the file to open; can include a path. If no name is specified, the Open dialog box displays.

Example

```
fileName$ = GetFileBox$("*.wsd", "Choose a File")
IF fileName$ <> "" THEN FileOpen fileName$
```

Asks the user for the name of a file to open and opens the selected file.

See Also

FileNew, FileNewDefaults

FileClose

Description

Closes the current document window. Optionally, closes all open documents.

Syntax

FileClose all%, noConfirm%

Parameters

all% - If TRUE, closes all documents. The default is FALSE.

noConfirm% - If TRUE, does not ask to save changes before closing. The default is FALSE.

Example

```
' Save and close the current document.
FileSave
FileClose 0, 0
```

Saves the current document and closes the window.

See Also

FileSave

FileImport

Description

Imports text, graphics, paragraph styles, frame styles, or EPS files.

Syntax

FileImport fileName$, filter$, reference%, options%

Parameters

fileName$ - The name of the file to import from.

filter$ - The name of a WSWin import filter; must match exactly the name of the filter in the Import dialog box.

reference% - The type of reference link, if any, to use for the import: 0 = no reference link, 1 = reference one-way, 2 = reference two-way.

options% - The import options to use for the import: 0 = use defaults, 1 = interpret format tags (used with text import), 2 = create new style when tabs change (used with text), 4 = overwrite duplicates (used with paragraph and frame styles), 8 = verify before overwriting (used with paragraph and frame styles), 16 = load function key assignments (used with paragraph and frame styles). You can add different options together if desired. The macro recognizes only those options that are valid for the type of import; all others will be ignored.

If no parameters are specified, displays the Import dialog box.

Example

```
FileImport "c:\wswin\template\default.wst", "WSWin Paragraph Styles",,
   4+16
FileImport "c:\wswin\template\default.wst", "WSWin Frame Styles",, 4+16
```

You can use these statement to update the paragraph and frame styles in the current document with updated styles from the default template or when you make a change to the default template.

See Also

FileExport

FileSave

Description

Saves the current document, or optionally, all open documents.

Syntax

FileSave all%

Parameters

all% - If TRUE, saves all open documents. The default is FALSE.

Example

```
' Save all documents and exit WSWin.
FileSave 1
FileExit
```

This macro saves all open documents and exits WSWin.

See Also

FileSaveAs, FileSaveAsCopy

FileSaveAs

Description

Saves the current document window under a new filename.

Syntax

FileSaveAs fileName$

Parameters

fileName$ - The new name to give to the file. If no name is specified, displays the Save As dialog box.

Example

```
' Saves the document under a new name based on the original
' Up to 9 copies of a file can be saved this way
orig$ = GetDocName$()
' Extract just the filename (no path or extension)
WHILE INSTR(orig$, "\") > 0
   path$ = path$ + LEFT$(orig$, INSTR(orig$, "\"))
   orig$ = MID$(orig$, INSTR(orig$, "\") + 1)
WEND
ext$ = MID$(orig$, INSTR(orig$, ".") + 1)
orig$ = LEFT$(orig$, INSTR(orig$, ".") - 1)
' See if there's already a number at the end of the filename
num% = VAL(RIGHT$(orig$, 1))
' Strip the number off the filename
IF num% > 0 THEN
   orig$ = LEFT$(orig$, LEN(orig$) - 1)
ELSE
   ' Make sure there are only seven characters
   orig$ = LEFT$(orig$, 7)
ENDIF
IF num% < 9 THEN
   fileName$ = path$ + orig$ + STR$(num% + 1) + "." + ext$
   FileSaveAs fileName$
ELSE
   MESSAGE "Can't save any more copies of the file."
ENDIF
```

This macro gets the name of the current file, saves the path and extension, and adds a number to the end of the file so that up to nine copies of the file can be saved automatically.

See Also

FileSave, FileSaveAsCopy

FileSaveAsCopy

Description

Saves the current document to a different file, leaving the current document window name unchanged. This statement is used if you want to save the file to a backup file, while you work with the original file.

Syntax

FileSaveAsCopy fileName$

Parameters

fileName$ - The new name for the file.

Example

```
orig$ = GetDocName$()
' Extract just the filename (no path)
WHILE INSTR(orig$, "\") > 0
   orig$ = MID$(orig$, INSTR(orig$, "\") + 1)
WEND
orig$ = "a:\" + orig$
FileSaveAsCopy orig$
```

This macro gets the current filename, removes the path information and saves the file to the root directory of drive A as a backup. The document window name will not change.

See Also

FileSave, FileSaveAs

FileExport

Description

Exports the contents of the current story line, selection, or frame.

Syntax

FileExport fileName$, filter$

Parameters

fileName$ - The name of the file to export to.

filter$ - The name of a WSWin export filter; must match exactly the name of the filter in the Import dialog box.

If no parameters are specified, displays the Export dialog box.

Example

```
FileExport "temp.ws7", "WordStar 7.0"
```

Exports the current story line to the file TEMP.WS7 in the current directory using the WordStar 7.0 export filter.

See Also

FileImport

FileRevertToSaved

Description

Discards all changes to the current document after the last save.

Syntax

FileRevertToSaved noConfirm%

Parameters

noConfirm% - If TRUE, no confirmation prompt appears. The default is FALSE.

Example

```
FileSave
TypeText "<Cr>Now is the time<Cr>"
FileRevertToSaved
```

Saves the current file, inserts some text and discards the changes.

See Also

FileClose, FileSave, FileSaveAs

FilePageSetup

Description

Changes page settings for the current document.

Syntax

FilePageSetup paperSize%, width%, height%, doubleSided%, notes%

Parameters

paperSize% - The size of the page: 0 = letter, 1 = legal, 2 = ledger, 3 = A3, 4 = A4, 5 = A5, 6 = B4, 7 = B5, 8 = Other. The default is letter.

width% - If paperSize% = 8, the width of the page in decipoints.

height% - If paperSize% = 8, the height of the page in decipoints.

doubleSided% - TRUE for double-sided pages. The default is FALSE.

notes% - The position of notes on the page: 0 for footnotes (bottom of page), 1 for endnotes (end of story). The default is 0.

If no parameters are specified, the Page Setup dialog box displays.

Example

```
FilePageSetup 1,,, 1, 1
```

Changes the page to legal size, double-sided, with endnotes.

See Also

FilePrintSetup

FilePrintSetup

Description

Changes the printer for the current document.

Syntax

FilePrintSetup printerName$, port$, setup%

Parameters

printerName$ - The name of the printer to use. The name must match exactly the name of the printer specified in the Windows Control Panel.

port$ - The port the printer is assigned to. A valid port must be specified. The specified printer must be available on the specified port for the routine to execute successfully.

setup% - If TRUE, displays the Setup dialog box associated with the specified printer, to allow the user to specify additional options for the printer.

If no parameters are specified, the Print Setup dialog box appears.

Example

```
FilePrintSetup "HP LaserJet III", "LPT1:", 1
```

Changes the printer for the document to the HP LaserJet III that is assigned to LPT1, and displays the Setup dialog box for the printer.

See Also

FilePrint, FilePageSetup

FilePrint

Description

Prints the current document according to the specified settings.

Syntax

FilePrint range%, from%, to%, copies%, draft%, collate%, reverse%, manual%, merge%

Parameters

range% - The range of pages to print: 0 = all, 1 = odd pages only, 2 = even pages only. The default is 0.

from% - If range% = 0, the first page to start printing from. The default is first page of document.

to% - If range% = 0, the last page number to print. The default is last page of document.

copies% - The number of copies to print. The default is 1.

draft% - If TRUE, use draft mode printing. The default is FALSE.

collate% - If TRUE, collate copies. The default is FALSE.

reverse% - If TRUE, print in reverse order. The default is FALSE.

manual% - If TRUE, use the manual feed of the printer. The default is FALSE.

merge% - If TRUE, interpret merge variables. The default is TRUE if a merge datafile is specified for the document. Otherwise, the setting is ignored.

If no parameters are specified, the Print dialog box appears.

Example

```
FilePrint 0, 1, 5, 2,,,,,
```

Prints two copies of pages 1 through 5 of the current document.

See Also

FilePrintDefaults

FilePrintDefaults

Description

Prints the current document using the default settings (one copy, all pages, no draft printing, no collating, normal order, not manual feed, merge variables only if a datafile is specified).

Syntax

FilePrintThe defaults

Example

```
FileSave
FilePrintDefaults
```

Saves the current document and prints it using the default settings.

See Also

FilePrint

FileExit

Description

Exits WSWin.

Syntax

FileExit noConfirm%

Parameters

noConfirm% - If TRUE, don't display a confirmation message about unsaved changes. The default is FALSE.

Example

```
FileSave 1
FileExit 1
```

Saves all open documents and exits without a confirmation message.

See Also

FileClose

Edit Statements

The Edit Statements section describes the available commands on the WSWin Edit menu. The Edit Statements include

- EditUndo
- EditCut
- EditCopy
- EditPaste
- EditPasteLink
- EditPasteSpecial
- EditDelete
- EditSelectAll
- EditFind
- EditFindNext
- EditReplace
- EditGoto
- EditGotoOffset
- EditObject
- EditLinks

EditUndo

Description

Reverses the most recent action.

Syntax

EditUndo

Example

```
TypeText "This is a test."
EditUndo
```

Inserts some text and uses Undo to remove the text.

See Also

EditPaste

EditCut

Description

Cuts the selected text or frame to the Clipboard.

Syntax

EditCut

Example

```
CharRight 1, 1
EditCut
CharRight 1
EditPaste
```

This macro reverses two characters by selecting and cutting the first one to the Clipboard and pasting it to the right of the next character.

See Also

EditCopy, EditPaste

EditCopy

Description

Copies the selected text or frame to the Clipboard.

Syntax

EditCopy

Example

```
WordRight 3, 1
EditCopy
ParaDown 1,
EditPaste
```

Selects the next three words, including the current word, and copies them to the Clipboard. The words are then pasted at the beginning of the next paragraph.

See Also

EditCut, EditPaste

EditPaste

Description

Pastes the contents of the Clipboard to the document.

Syntax

EditPaste

Example

See EditCut and EditCopy.

See Also

EditPasteLink, EditPasteSpecial

EditPasteLink

Description

Pastes the contents of the Clipboard to the document and establishes a link to the original document where the Clipboard contents were first created.

Syntax

EditPasteLink

Example

```
' Perform some work in another application and copy it
' to the Clipboard
EditPasteLink
```

Creates a link between the original application that created some data and the current document.

See Also

EditPaste, EditPasteSpecial

EditPasteSpecial

Description

Pastes the contents of the Clipboard to the document, using a specified format.

Syntax

EditPasteSpecial format%, link%

Parameters

format% - The Clipboard format to use when pasting. The default is 1, the first format in the list.

link% - If TRUE, then establish a link with the original application that created the data on the Clipboard. The default is FALSE.

If no parameters are specified, the Paste Special dialog box appears.

Example

```
' We know that the Clipboard contains a bitmap from
' Paintbrush.  We want to paste it as a regular bitmap
' rather than an OLE object.
EditPasteSpecial 2, 0
```

Paintbrush also places formats for "picture" and "bitmap" on the Clipboard. The bitmap format is the second format in the list and is used to paste an object in a graphic frame. (If "paintbrush picture object" is used to paste in a graphic frame, a new OLE frame is created inside the graphic frame.)

See Also

EditPaste, EditPasteLink

EditDelete

Description

Deletes the specified text or frame.

Syntax

EditDelete type%, confirm%

Parameters

type% - The type of object to delete: 0 = delete the character to the right of the insertion point or the current selection, 1 = delete the word to the right of the insertion point, 2 = delete to the end of the line, 3 = delete the previous character, 4 = delete previous word, 5 = delete to the beginning of the line, 6 = delete the current frame.

confirm% - If TRUE, display a confirmation message before deleting the frame. The default is FALSE. This parameter is valid only if type% = 6.

Example

```
WordRight 3, 1
EditDelete ,
LineDown 1,
EditDelete 2,
```

Selects the next three words and deletes them, then goes to the next line and deletes to the end of the line.

See Also

EditCut

EditSelectAll

Description

Selects the entire story or frame contents.

Syntax

EditSelectAll

Example

```
EditSelectAll
EditCut
```

Selects the entire story and cuts it to the Clipboard.

See Also

EditCut, EditCopy, Navigation Statements

EditFind

Description

Finds the specified text in the document.

Syntax

ret% = EditFind (findText$, direction%, findBy%, start%, ignoreCase%, ignoreTags%, select%)

Parameters

findText$ - The text to find. Use WSWin text conventions.

direction% - The direction of the search: 0 = forward, 1 = backward. The default is forward.

findBy% - How to search for text: 0 = search by story line, 1 = search by page. The default is 0.

start% - Where to begin the search: 0 = current location, 1 = beginning of document. The default is 0. (When you search by page and specify the current location, the search starts at the beginning of the current page. When you search by story line and specify the current location, the search starts at the insertion point.)

ignoreCase% - Whether to match the case of the findText exactly. The default is TRUE.

ignoreTags% - Whether to ignore any tags except those specified in findText$. The default is TRUE.

select% - If TRUE and the text is found, extend the selection to the end of the found text. The default is FALSE.

If no parameters are specified, the Find dialog box appears.

Returns

TRUE if the text is found.

Example

```
IF EditFind("test",,,,, ) THEN MESSAGE "Text was found."
```

If the specified text is found, a message appears. When this statement executes and if the text is found, it will be selected after the statement ends.

See Also

EditFindNext, EditReplace

EditFindNext

Description

Repeats the search specified in the previous EditFind statement.

Syntax

ret% = EditFindNext (select%)

Parameters

select% - If TRUE and the text is found, extend the selection to the end of the found text. The default is FALSE.

Returns

TRUE if the text is found.

Example

```
ret% = EditFind(".",,,,, )
ret% = EditFindNext(1)
```

Searches for the next period character and selects the text to the next occurrence of a period.

See Also

EditFind, EditReplace

EditReplace

Description

Replaces one text string with another.

Syntax

ret% = EditReplace (findText$, replaceText$, direction%, findBy%, start%, ignoreCase%, ignoreTags%, viewContext%, maintainCase%, confirm%)

Parameters

findText$ - The text to find. Use WSWin text conventions.

replaceText$ - The text to replace with. Use WSWin text conventions.

direction% - The direction of the search: 0 = forward, 1 = backward. The default is 0.

findBy% - How to search for text: 0 = search by story line, 1 = search by page. The default is 0.

start% - Where to begin the search: 0 = current location, 1 = beginning of document. The default is 0. (When you search by page and specify the current location, the search starts at the beginning of the current page. When you search by story line and specify the current location, the search starts at the insertion point.)

ignoreCase% - Whether to match the case of the findText exactly. The default is TRUE.

ignoreTags% - Whether to ignore any tags except those specified in findText$. The default is TRUE.

viewContext% - Whether to display the context in the document as the text is found. The default is TRUE.

maintainCase% - Whether to maintain the case of the findText$ and replaceText$. The default is FALSE. This parameter is valid only if ignoreCase% is TRUE.

confirm% - If TRUE, waits for confirmation before performing the replace. The default is TRUE.

Returns

TRUE if the specified text was replaced at least once.

Example

```
IF EditReplace("this", "that",,,,,, 0,, 0) THEN
   MESSAGE "Text was replaced."
ENDIF
```

Replaces all occurrences of the word "this" with "that" without showing the context or asking for confirmation. If successful, a message appears.

See Also

EditFind

EditGoto

Description

Goes to a specific location in the document.

Syntax

EditGoto where%, num%, bookmark$, select%

Parameters

where% - The location to go to: 0 = specific page, 1 = first page, 2 = last page, 3 = previous page, 4 = next page, 5 = current header, 6 = current footer, 7 = bookmark, 8 = footnote, 9 = frame. The default is 0.

num% - If where% is 0, the number of the page to go to. If where% is 7, the number of the frame to go to.

bookmark$ - If where% is 7, the name of the bookmark to go to.

select% - If TRUE, extend the selection to the new location. The selection will be extended only if it is valid in the new context. The default is FALSE.

If no parameters are specified, the Go To dialog box appears.

Example

```
EditGoto 0, 4,,
```

Go to page 4.

```
EditGoto 7,, "Save",
```

Go to a bookmark called "Save"

```
frameID% = InsertFrame(0, 720, 720, 1440, 1440, "Text", 1)
ParentFrame
EditGoto 9, frameID%,,
```

Create a text frame, go to the parent frame, and go back to the newly-created frame.

See Also

EditGotoOffset

EditGotoOffset

Description

Go a specific location in the current story line.

Syntax

EditGotoOffset where%, select%

Parameters

where% - The location to go to. This value is always relative to the beginning of the story line.

select% - If TRUE, extend the selection to the new location. The default is FALSE.

Example

```
ret% = GetTextOffset(start%, end%)
StartOfStory
EditGotoOffset start%,
IF ret% THEN EditGotoOffset end%, 1
```

Gets the text offsets of the current selection and goes to the beginning of the story line. Returns to the starting point of the previous selection, and if text was selected, reselects the text to the ending position.

See Also

EditGoto, GetTextOffset

EditObject

Description

Edits the current OLE object.

Syntax

EditObject action%

Parameters

action% - The action to perform on the OLE object. This value corresponds to the verbs associated with the OLE object that appear in the submenu when the Edit•Object command is selected. The default is 1, the first verb in the list.

Example

```
NextFrame
IF GetFrameType() = 4 THEN EditObject 1
```

Goes to the next frame and, if the frame is an OLE frame, performs the first edit verb on the selected frame.

See Also

InsertObject

EditLinks

Description

Displays the Links dialog box to manage DDE and OLE links.

Syntax

EditLinks

Example

```
ret% = InsertObject("Paintbrush Picture Object", "c:\windows\arches.bmp",
   1, 0, 0, 720, 720)
EditLinks
```

Inserts an OLE object linked to the arches.bmp bitmap, one inch from the top and left edges of the current frame. Then, displays the Links dialog box to verify that the object was linked.

See Also

EditPasteLink, EditPasteSpecial

View Statements

The View Statements section describes the available commands on the WSWin View menu. The View Statements include

- ViewEditor
- ViewDraftTags
- ViewZoomLevel
- ViewFacingPages
- ViewDisplay
- ViewTagList
- ViewRedrawWindow
- ViewEditMode
- ViewFreezeScreen

ViewEditor

Description

Switches from the Page Editor to the Draft Editor and back again.

Syntax

ViewEditor page%

Parameters

page% - If TRUE, switch to the Page Editor. If FALSE, switch to the Draft Editor. If not specified, switch to the other editor.

Example

```
ViewEditor 1
ViewEditor 0
ViewEditor
```

First, switch to the Page Editor. Next, switch to the Draft Editor. Finally, switch back to the Page Editor.

See Also

ViewDraftTags, ViewEditMode, GetEditor

ViewDraftTags

Description

Switches the display of tags in the Draft Editor.

Syntax

ViewDraftTags on%

Parameters

on% - If TRUE, turns tags on in the Draft Editor. If FALSE, turns Draft Editor tags off. If not specified, toggles the display of the tags.

Example

```
ViewEditor 0
ViewDraftTags 1
```

Switches to the Draft Editor and turns Draft Editor tags on.

See Also

ViewEditor

ViewZoomLevel

Description

Changes the zoom level in the current document.

Syntax

ViewZoomLevel percent%

Parameters

percent% - Sets the zoom percentage in the document from 25 to 200. Other valid options: if set to 0, uses the default zoom level stored in the WSW.INI file; if 1, set to 100%; if 2, fit the page in the window; if 3, zoom in by 25%; if 4, zoom out by 25%. If not specified, display the Zoom Level dialog box.

Example

```
ViewZoomLevel 2
ViewZoomLevel 3
ViewZoomLevel 150
```

First, set the zoom level to fit the page in the window. Next, zoom in by 25%. Finally, zoom to 150%.

See Also

ViewFacingPages

ViewFacingPages

Description

Turns the display of facing pages on and off for the current document.

Syntax

ViewFacingPages facing%

Parameters

facing% - If TRUE, turns facing pages on. If FALSE, turns facing pages off. If not specified, toggles the display of facing pages.

Example

```
ViewZoomLevel 2
ViewFacingPages 1
```

Sets the zoom level to fit the page in the document window and turns on facing pages.

See Also

ViewZoomLevel

ViewDisplay

Description

Turns the display of various screen elements on and off.

Syntax

ViewDisplay ruler%, Toolbox%, styleBar%, frameBar%, statusBar%, textOnly%, frameOutlines%, paraSymbols%

Parameters

ruler% - If TRUE, turns the ruler on. If FALSE, turns the ruler off.

Toolbox% - If TRUE, turns the Toolbox on. If FALSE, turns the Toolbox off.

styleBar% - If TRUE, turns the Style Bar on. If FALSE, turns the Style Bar off.

frameBar% - If TRUE, turns the Frame Bar on. If FALSE, turns the Frame Bar off.

statusBar% - If TRUE, turns the Status Bar on. If FALSE, turns the Status Bar off.

textOnly% - If TRUE, turns text only on (hiding table, graphic, EPS, and OLE frames). If FALSE, turns text only off.

frameOutlines% - If TRUE, turns frame outlines on. If FALSE, turns frame outlines off.

paraSymbols% - If TRUE, turns paragraph symbols on. If FALSE, turns paragraph symbols off.

For each parameter, the default is no change to the setting.

Example

```
ViewDisplay 1,,, 1,,, 1, 0
```

Turns the ruler, Frame Bar, and frame outlines on, and turns off paragraph symbols.

See Also

ViewTagList

ViewTagList

Description

Turns the tag list window on and off.

Syntax

ViewTagList on%

Parameters

on% - If TRUE, turns the tag list window on. If FALSE, hides the window. If not specified, toggles the display of the tag list window.

Example

```
ViewEditor 1
ViewTagList 1
```

Switches to the Page Editor and turns on the tag list window.

See Also

ViewEditor, ViewDisplay

ViewRedrawWindow

Description

Redraws the current document window.

Syntax

ViewRedrawWindow

Example

```
frame% = InsertFrame(0, 720, 720, 720, 720, "Text", 1)
ViewRedrawWindow
```

Inserts a frame in the current frame and redraws the window.

See Also

ViewZoomLevel

ViewEditMode

Description

Switches between frame mode and edit (content) mode.

Syntax

ViewEditMode mode%

Parameters

mode% - The mode setting to switch to: 0 = toggle between the two, 1 = edit mode, 2 = frame mode. The default is 0.

Returns

The previous mode setting.

Example

```
oldMode% = ViewEditMode(2)
NextFrame
GetFrameSize x%, y%, w%, h%
FrameSize x%, y%, w%, h% + 720
ret% = ViewEditMode(oldMode%)
```

Switches to frame mode, goes to the next frame, gets the size of the frame and adds one inch to the frame height, and restores the original edit mode.

See Also

ViewEditor

ViewFreezeScreen

Description

Prevents the screen from repainting during the execution of a macro. This is used to allow macros to modify the document without excessive screen redrawing.

Syntax

old% = ViewFreezeScreen (freeze%)

Parameters

freeze% - If TRUE, freeze the screen so that no redrawing takes place. If FALSE, allow screen redrawing. If no parameter is specified, then no action is taken, but the function still returns the current state.

Returns

The current freeze state of the screen.

Example

```
old% = ViewFreezeScreen(1)
frame% = InsertFrame(0, 720, 720, 720, 720, "Text", 1)
ParentFrame
frame2% = InsertFrame(0, 1440, 1440, 720, 720, "Text", 1)
ParentFrame
TypeText "This is a test."
old% = ViewFreezeScreen(old%)
```

Freezes the screen, inserts two frames, and types some text in the page frame. The screen is then released so that all three elements can be displayed. Until the last statement, no frames or text will appear.

See Also

ViewRedrawWindow

Related Topics

Improving Macro Running Speed

Insert Statements

The Insert Statements section describes the available commands on the WSWin Insert menu. The Insert Statements include

- InsertAutoNumber
- InsertPageBreak
- InsertColumnBreak
- InsertTag
- InsertKerningTag
- InsertDate
- InsertTime
- InsertFileName
- InsertFrame
- InsertNote
- InsertHeader
- InsertFooter
- InsertCaption
- InsertPage
- InsertListEntry
- InsertIndexEntry
- InsertBookmark
- InsertReference
- InsertGuideWord
- InsertObject

InsertAutoNumber

Description

Inserts an auto number or an auto number reset tag at the insertion point.

Syntax

InsertAutoNumber numberStyle$, level%, reset%, startNumber%, insertNumber%

Parameters

numberStyle$ - The name of the auto number style to use. If not specified, the Insert Auto Number dialog box appears.

level% - If using the Outline number style, the outline level for the number.

reset% - If TRUE, reset the numbering with the specified starting number. The default is FALSE. If the Oultine number style is reset, then the level% parameter is ignored.

startNumber% - The starting number for the auto number, if reset% is TRUE.

insertNumber% - If TRUE, insert the auto number after the reset tag. This parameter is ignored if reset% is FALSE. The default is FALSE.

Example

```
InsertAutoNumber "Outline", 3,,,
InsertAutoNumber "Page",, 1, 3,
```

Inserts the next auto number for level 3 of the Outline number style and inserts a reset tag for the Page number style, resetting the page numbering to 3.

See Also

InsertHeader, InsertFooter

InsertPageBreak

Description

Inserts a page break at the insertion point.

Syntax

InsertPageBreak

Example

```
TypeText "This is a test."
InsertPageBreak
TypeText "This text is on the next page."
```

Inserts some text in the document and inserts a page break to go to another page before entering more text.

See Also

InsertColumnBreak

InsertColumnBreak

Description

Inserts a column break at the current insertion point. If the text is already in the last column on the page, or if there is only one column on the page, a new page is created.

Syntax

InsertColumnBreak

Example

```
DefineFrameStyle ,
DefineFrameColumns 2,,,,
EndDefineFrameStyle
TypeText "This is some text."
InsertColumnBreak
TypeText "This text is in column 2."
```

Changes the current frame to two columns, then inserts some text in column 1 followed by a column break. The remaining text is inserted in column 2.

See Also

InsertPageBreak

InsertTag

Description

Inserts the specified tag at the insertion point.

Syntax

InsertTag tag%

Parameters

tag% - The tag to insert: 1 = nonbreaking hyphen, 2 = optional hyphen, 3 = em space, 4 = en space, 5 = figure space, 6 = thin space, 7 = decimal separator, 8 = thousands separator, 9 = footnote break.

Example

```
TypeText "This is a test."
InsertTag 3
TypeText "There is an em space between these two sentences."
```

After typing the first sentence, an em space is inserted before the second sentence.

See Also

TypeText

InsertKerningTag

Description

Adjusts the space between two characters or the space between all characters in selected text.

Syntax

InsertKerningTag value%

Parameters

value% - The amount, in 1/1000s of an em, to kern the text. A negative value is used to decrease the space between characters; a positive value increases the space between characters. The default is 83.

Example

```
IF GetPrevChar$ = "V" and GetNextChar$ = "A" THEN
   InsertKerningTag -120
ENDIF
StartOfLine
EndOfLine 1
InsertKerningTag 100
```

In the first section, a kerning tag with a value of -120 is inserted between the characters VA. In the second section, the entire line is selected and the spacing between all the characters in the line is increased by 10% (100/1000).

See Also

InsertTag, TypeText

InsertDate

Description

Inserts the current date at the insertion point, either as a system variable or as regular text.

Syntax

InsertDate long%, asText%

Parameters

long% - If TRUE, use the long date format. The default is TRUE.

asText% - If TRUE, inserts the date as a text string. If FALSE, inserts the date as a system variable that is interpreted when the document is printed. The default is FALSE.

Example

```
TypeText "Document created on:<Tb>"
InsertDate 0, 1
TypeText "<Tb>"
InsertTime 1
TypeText "<Cr>Document printed on:<Tb>"
InsertDate 1,
TypeText "<Tb>"
```

Inserts the text on the first line, followed by the current date in short form and the current time, both as text. On the next line, inserts text to show when the document is printed, followed by a date variable and a time variable that will expand to the current date and time when the document is printed.

Note The Control Panel Date/Time settings determine the long and short formats for the date.

See Also

InsertTime, InsertFilename

InsertTime

Description

Inserts the current time at the insertion point, either as a system variable or as regular text.

Syntax

InsertTime asText%

Parameters

asText% - If TRUE, inserts the time as a text string. If FALSE, inserts the time as a system variable that is interpreted when the document is printed. The default is FALSE.

Example

See InsertDate.

Note The Control Panel Date/Time settings determine the format for the time.

See Also

InsertFilename

InsertFileName

Description

Inserts the current filename at the insertion point, either as a system variable or as regular text.

Syntax

InsertFilename path%, upperCase%, asText%

Parameters

path% - If TRUE, include the path with the filename. The default is FALSE.

upperCase% - If TRUE, force the filename to be uppercase. The default is FALSE.

asText% - If TRUE, inserts the filename as a text string. If FALSE, inserts the filename as a system variable that is interpreted when the document is printed. The default is FALSE.

Example

```
TypeText "Original filename:<Tb>"
InsertFilename 1, 0, 1
TypeText "<Cr>Current filename:<Tb>"
InsertFilename 1, 0, 0
```

Inserts the current filename as text, followed by a variable that expands to the current filename when the document is printed. If the name of the file is changed before the document is printed, the two filenames will be different.

See Also

InsertDate, InsertTime

InsertFrame

Description

Inserts a frame in the current frame.

Syntax

frameNum% = InsertFrame (type%, originX%, originY%, width%, height%, frameStyle$, select%)

Parameters

type% - The type of frame to insert: 0 = text frame, 1 = table, 2 = graphics, 3 = EPS. OLE frames are created automatically, not with this function. The default is 0.

originX% - The horizontal starting position for the frame in decipoints, relative to the left edge of the parent frame.

originY% - The vertical starting position for the frame in decipoints, relative to the top edge of the parent frame.

width% - The width of the frame in decipoints.

height% - The height of the frame in decipoints.

frameStyle$ - The frame style to apply to the new frame. The default is the default frame style for the specified type.

select% - If TRUE, select the frame and switch to edit mode. If FALSE, frame mode remains active, and the new frame is not active. The default is TRUE.

Returns

The frame number of the new frame. If no frame could be inserted, the function returns -1.

Example

```
frame% = InsertFrame(0, 720, 720, 1440, 1440,, )
```

See Also

EditGoto, FrameSize, GetFrame, CountFrames

InsertNote

Description

Inserts a footnote or endnote at the insertion point.

Syntax

InsertNote noteText$

Parameters

noteText$ - The text for the footnote or endnote. Use WSWin text conventions.

If no text is specified, the footnote is inserted, and the next text will be inserted as part of the footnote. If no text is specified and the insertion point is already in a footnote, the insertion point returns to the document immediately after the footnote. If text is specified, then the footnote is inserted and the insertion point remains in the document immediately after the footnote.

Example

```
InsertNote
TypeText "This text is part of the footnote."
InsertNote
TypeText "This text is in the body of the document."
InsertNote "This is another footnote."
TypeText "This is more text in the document."
```

Two footnotes are inserted in the document. The first note uses the regular TypeText command. The second footnote uses the InsertNote command for the footnote text.

Note To change footnotes to endnotes and vice versa, use the FilePageSetupcommand.

See Also

TypeText

InsertHeader

Description

Inserts a header on the current page.

Syntax

frameNum% = InsertHeader (text$, confirm%)

Parameters

text$ - The text to insert in the header. If no text is specified, the header remains empty.

confirm% - If TRUE, a message indicates that a header already flows on the page (i.e., a previous page has a header). If FALSE, no message appears and the header is inserted. The default is TRUE.

Returns

The frame number of the new header frame. If no header frame is created because a header frame already exists on the page, the function returns -1.

Example

```
frame% = InsertHeader ("Header Text", 0)
IF frame% = 0 THEN MESSAGE "Couldn't create header."
```

Inserts a header with the string Header Text in it. No confirmation is given if a header already flows on the page.

See Also

InsertFooter, EditGoto

InsertFooter

Description

Inserts a footer on the current page.

Syntax

frameNum% = InsertFooter (text$, confirm%)

Parameters

text$ - The text to insert in the footer. If no text is specified, the footer remains empty.

confirm% - If TRUE, a message indicates that a footer already flows on the page (i.e., a previous page has a footer). If FALSE, no message appears and the footer is inserted. The default is TRUE.

Returns

The frame number of the new footer frame. If no footer frame is created because a footer frame already exists on the page, the function returns -1.

Example

```
frame% = InsertFooter ("Footer Text", 0)
IF frame% = 0 THEN MESSAGE "Couldn't create footer."
```

Inserts a footer with the string Footer Text in it. No confirmation is given if a footer already flows on the page.

See Also

InsertHeader, EditGoto

InsertCaption

Description

Inserts a caption frame on to the current frame.

Syntax

frameNum% = InsertCaption (position%, text$)

Parameters

position% - The position of the caption frame: 0 = bottom, 1 = top, 2 = left, 3 = right. The default is 0.

text$ - The text to be inserted in the caption frame. If no text is specified, the frame remains empty.

Returns

The frame number of the new caption frame. If a caption couldn't be inserted because the current frame does not support a caption frame, the function returns -1.

Example

```
frame% = InsertFrame(0, 720, 720, 1440, 1440, "Text", 1)
caption% = InsertCaption(0, "This is a bottom caption.")
```

Inserts a frame and inserts a caption below the frame with text in the caption.

See Also

InsertFrame

InsertPage

Description

Inserts one or more pages in the document.

Syntax

frameNum% = InsertPage (styleName$, position%, numPages%, link%)

Parameters

styleName$ - The frame style to apply to the new pages. The default is "PageText".

position% - Where the pages should be inserted: 1 = after current page, 2 = before current page, 3 = at end of document. The default is 3.

numPages% - The number of pages to insert. The default is 1. Maximum is 99.

link% - How the pages should be linked to the current page: 0 = don't link pages, 1 = link pages to current frame, 2 = link pages to subsequent frame, 3 = link pages to both current and subsequent frames. The default is 1.

If no parameters are specified, the Insert Page dialog box appears.

Returns

The frame number of the first new page. If no page frame could be inserted, the function returns -1.

Example

```
frame% = InsertPage ("PageText", 1, 1, 3)
```

Inserts a page after the current page, linked to both the current and subsequent page frames.

Note Auto Pagination must be off to use this command. If Auto Pagination is not off, the command will not execute.

See Also

EditGoto

InsertListEntry

Description

Inserts a list entry at the insertion point.

Syntax

InsertListEntry listName$, text$, visible%

Parameters

listName$ - The name of the list you want to use.

text$ - The text to insert as the list entry. Use WSWin text conventions.

visible% - If TRUE, the list entry will be visible. If FALSE, the list entry will not be visible. The default is FALSE.

If no parameters are specified, the List Entry dialog box appears.

Example

```
InsertListEntry "Annotation", "This is some annotation text.", 1
```

Inserts a visible entry for the Annotation list.

See Also

InsertIndexEntry, InsertBookmark, ToolsBuildList

InsertIndexEntry

Description

Inserts an index entry at the insertion point.

Syntax

InsertIndexEntry text1$, text2$, text3$, sort1$, sort2$, sort3$, visible%, includePage%

Parameters

text1$, text2$, text3$ - The text for level 1, 2, and 3 of the index entry, respectively. If text2$ is specified, text1$ must also be specified. If text3$ is specified, then both text2$ and text1$ must be specified. Use WSWin text conventions to format the entries.

sort1$, sort2$, sort3$ - The text to use when sorting the entries.

visible% - If TRUE, the index entry will be visible. The default is FALSE.

includePage% - If TRUE, the page number will be included with the entry. This setting applies to all index entries in the entire document; the last setting used will apply to all entries. The default is TRUE.

If no parameters are specified, the Index Entry dialog box appears.

Example

```
InsertIndexEntry "cars",,,,,,,
InsertIndexEntry "cars", "Chevrolet", "<It 1>Impala<It 0>",,,,,
InsertIndexEntry "cars", "Pontiac",,,,,,
```

Inserts three index entries: a one-level entry for "cars", a three-level entry for the Chevrolet Impala, and a two-level entry for the Pontiac. Note the use of Italics for the word Impala.

See Also

InsertListEntry, ToolsBuildIndex

InsertBookmark

Description

Inserts a bookmark at the insertion point.

Syntax

InsertBookmark name$, autoNum$, level%, text$, delete%

Parameters

name$ - The name of the bookmark. If a bookmark with that name already exists, it will be deleted and a new one inserted at the current insertion point.

autoNum$ - The auto number style to use for the bookmark, if any.

level% - If the Outline number style is used, the outline level for the auto number.

text$ - The text to use for the bookmark, if any. You can use WSWin text conventions to format the text.

delete% - If TRUE, the bookmark will be deleted. The default is FALSE.

If no parameters are specified, the Bookmark dialog box appears.

Example

```
InsertBookmark "SavePosition",,,,
EndOfStory
EditGoto 7,, "SavePosition",
InsertBookmark "SavePosition",,,, 1
```

Inserts a bookmark called SavePosition, goes to the end of the story, goes back to the bookmark and deletes it.

Note The autoNum$, level%, and text$ parameters are most useful when you want to refer to bookmarks with InsertReference elsewhere in the document. If you want to use bookmarks as placeholders that you can go to later, you do not need to specify these parameters.

See Also

InsertReference, EditGoto

InsertReference

Description

Inserts a reference to a bookmark at the insertion point.

Syntax

InsertReference name$, type%

Parameters

name$ - The name of the bookmark.

type% - The type of reference to the bookmark: 0 = number of the page where the bookmark is, 1 = the text of the bookmark (specified in the text$ parameter), 2 = the auto number of the bookmark. The default is 0.

Example

```
InsertBookmark "Fig1", "Figure",, "<It 1>Sales Growth<It 0>",
PageDown
ParaDown 2
TypeText "Refer to figure "
InsertReference "Fig1", 2
TypeText ", " + CHR$(34)
InsertReference "Fig1", 1
TypeText "," + CHR$(34) + " on page "
InsertReference "Fig1", 0
TypeText "."
```

Inserts a bookmark, called Fig1, for a figure in the document. The bookmark uses the Figure auto number style. If the figure displays the number 4 and appears on page 32, the references and text inserted on the next page read as follows:

> Refer to figure 4, "*Sales Growth,*" on page 32.

The example uses all three references to the bookmark.

See Also

InsertBookmark

InsertGuideWord

Description

Inserts a guide word in the header or footer at the insertion point.

Syntax

InsertGuideWord from%, styleName$

Parameters

from% - Which occurrence of the style to use for the guide word: 1 = first occurrence, 2 = last occurrence.

styleName$ - The style name to use for the guide word

If no parameters are specified, the Guide Words dialog box appears.

Example

```
frame% = InsertHeader( , 1)
InsertGuideWord 1, "Heading"
TypeText " - "
InsertGuideWord 2, "Heading"
```

Inserts a header on the current page, and inserts a guide word for the first occurrence of the style Heading, followed by a dash and another guide word for the last occurrence of the style Heading. If the headings on that page include "Chevrolet", "Pontiac", "Cadillac", and "Chrysler", the guide words in the header would read: Chevrolet - Chrysler.

Note You can insert guide words only in headers and footers.

See Also

InsertHeader, InsertFooter

InsertObject

Description

Inserts an OLE object in the document.

Syntax

frameNum% = InsertObject (objectType$, fileName$, link%, package%, invokeServer%, xPos%, yPos%)

Parameters

objectType$ - The name of the object to insert. It must match exactly a name in the Windows registration data. Look at the entries in the Insert Object dialog box for the names.

fileName$ - The name of a file that contains the object you want to insert.

link% - If TRUE, insert as a linked object. The default is FALSE.

package% - If TRUE, insert as a packaged object. The default is FALSE.

invokeServer% - If TRUE, run the server for the object type. The default is FALSE if a filename is specified and TRUE if no filename is specified.

xPos% - The horizontal position of the OLE frame in decipoints, relative to the left edge of the parent frame. The default is the location of the insertion point.

yPos% - The vertical position of the OLE frame in decipoints, relative to the top edge of the parent frame. The default is the location of the insertion point.

If no parameters are specified, the Insert Object dialog box appears.

Returns

The frame number of the new OLE frame containing the object. If the frame was not inserted, the function returns -1.

Example

```
frame% = InsertObject("Paintbrush Picture",,,, 720, 720)
```

Inserts a paintbrush picture object one inch from the top and left edges of the page frame. Because no filename was specified, Paintbrush will start so that a new object can be created.

See Also

EditObject, EditLinks

Style Statements

The Style Statements section describes the available commands on the WSWin Style menu. The Style Statements include

- StyleApplyStyle
- StyleApplyFont
- StylePointSize
- StylePlain
- StyleBold
- StyleItalic
- StyleUnderline
- StyleStrikeout
- StyleSuperSub
- StyleLowercase
- StyleUppercase
- StyleTitlecase
- StyleTextColor
- StyleAlign
- StyleModifyStyle
- StyleCreateStyle
- StyleDeleteStyle
- StyleCreateLine
- StyleDeleteLine
- StyleKeys
- StyleAutoNumModify
- StyleAutoNumCreate
- StyleAutoNumDelete
- StyleFootnote
- DefineStyle
- DefineStyleNextStyle
- DefineStyleFont
- DefineStyleParagraph
- DefineStyleKerning
- DefineStyleTabSet
- DefineStyleTabClear
- DefineStyleTabClearAll
- DefineStyleColor
- DefineStyleLines
- DefineStyleBullet
- DefineStyleNumbering
- DefineStyleDropCap
- DefineStyleSuperSub
- DefineStyleWordSpacing
- EndDefineStyle

StyleApplyStyle

Description

Applies the specified paragraph style to the current paragraph or text selection.

Syntax

StyleApplyStyle styleName$, selection%

Parameters

styleName$ - The name of the paragraph style to apply. If not specified, displays the Apply Paragraph Style dialog box.

selection% - If TRUE, apply the style to the selected text only (i.e., as a character style). If FALSE, applies the style to each paragraph in the selection. The default is contained in the WSW.INI file, and is set in the Preferences dialog box.

Example

```
StyleApplyStyle "BodyText",
StyleApplyStyle style$, 1
```

The first line applies the BodyText style to the current paragraph. The second line applies the style contained in the variable style$ to every paragraph in the current selection.

See Also

StyleApplyFont, StyleModifyStyle

StyleApplyFont

Description

Applies the specified font to the current selection or to new text entered at the insertion point.

Syntax

StyleApplyFont fontName$

Parameters

fontName$ - The name of the font to apply. If not specified, the Apply Font dialog box appears.

Example

```
WordRight 3, 1
StyleApplyFont "Arial"
CharRight 1,,
StyleApplyFont font$
TypeText "This font is different."
```

In the second line, the font Arial is applied to the three words selected in the first line. In the fourth line, the font contained in the variable font$ is applied to the text entered in the last line.

See Also

StyleApplyStyle, StylePointSize, StylePlain, StyleBold, StyleItalic, StyleUnderline, StyleStrikeout, StyleSuperSub, StyleTextColor

StylePointSize

Description

Applies the specified point size to the selected text or to new text entered at the insertion point.

Syntax

StylePointSize size%

Parameters

size% - The size, in decipoints, to apply to the selected text or to new text. If not specified, the Point Size dialog box appears.

Example

```
WordRight 3, 1
StylePointSize 100
StylePointSize size%
TypeText "This text is in a different size."
```

Three words are selected and changed to a 10-point font. Then the size changes to the value in the variable size%, and the new text is entered in that size.

See Also

StyleApplyStyle, StyleApplyFont, StylePlain, StyleBold, StyleItalic, StyleUnderline, StyleStrikeout, StyleSuperSub, StyleTextColor

StylePlain

Description

Removes all bold, italic, underline, strikeout, and superscript/subscript tags from the selected text or to new text entered at the insertion point.

Syntax

StylePlain

Example

```
StyleBold 1
StyleItalic 1
TypeText "This text is bold and italic."
StylePlain
TypeText "This text is not bold or italic."
```

See Also

StyleApplyStyle, StyleApplyFont, StylePointSize, StyleBold, StyleItalic, StyleUnderline, StyleStrikeout, StyleSuperSub, StyleTextColor, GetStyleFont, DefineStyleFont

StyleBold

Description

Toggles the bold attribute on or off for selected text or to new text entered at the insertion point.

Syntax

StyleBold status%

Parameters

status% - The bold setting to apply: 0 = off, 1 = on, 2 = toggle, 3 = return to style. The default is 2.

Example

```
StyleBold 1
TypeText "This text is bold."
StyleBold 0
TypeText "This text is not bold."
```

See Also

StyleApplyStyle, StyleApplyFont, StylePointSize, StylePlain, StyleItalic, StyleUnderline, StyleStrikeout, StyleSuperSub, StyleTextColor, GetStyleFont, DefineStyleFont

StyleItalic

Description

Toggles the italic attribute on or off for selected text or to new text entered at the insertion point.

Syntax

StyleItalic status%

Parameters

status% - The italic setting to apply: 0 = off, 1 = on, 2 = toggle, 3 = return to style. The default is 2.

Example

```
StyleItalic 1
TypeText "This text is italic."
StyleItalic 0
TypeText "This text is not italic."
```

See Also

StyleApplyStyle, StyleApplyFont, StylePointSize, StylePlain, StyleBold, StyleUnderline, StyleStrikeout, StyleSuperSub, StyleTextColor, GetStyleFont, DefineStyleFont

StyleUnderline

Description

Toggles the underline attribute on or off for selected text or to new text entered at the insertion point.

Syntax

StyleUnderline status%

Parameters

status% - The underline setting to apply: 0 = off, 1 = single, 2 = double, 3 = return to style. The default is 1 if underline is off, 3 if underline is on.

Example

```
StyleUnderline 1
TypeText "This text is single underlined."
StyleUnderline 0
TypeText "This text is not single underlined."
```

See Also

StyleApplyStyle, StyleApplyFont, StylePointSize, StylePlain, StyleBold, StyleItalic, StyleStrikeout, StyleSuperSub, StyleTextColor, GetStyleFont, DefineStyleFont

StyleStrikeout

Description

Toggles the strikeout attribute on or off for selected text or to new text entered at the insertion point. Use the DefineStyleFont statement to change the strikeout character.

Syntax

StyleStrikeout status%

Parameters

status% - The strikeout setting to apply: 0 = off, 1 = on, 2 = toggle, 3 = return to style. The default is 2.

Example

```
StyleStrikeout 1
TypeText "This text is struck out."
StyleStrikeout 0
TypeText "This text is not struck out."
```

See Also

StyleApplyStyle, StyleApplyFont, StylePointSize, StylePlain, StyleBold, StyleItalic, StyleUnderline, StyleSuperSub, StyleTextColor, GetStyleFont, DefineStyleFont

StyleSuperSub

Description

Toggles the superscript or subscript attribute for selected text.

Syntax

StyleSuperSub status%

Parameters

status% - The super/subscript setting to apply: 0 = off, 1 = superscript, 2 = subscript, 3 = return to style. The default is 1.

Example

```
TypeText "Einstein's theory of relativity is e=mc"
StyleSuperSub 1
TypeText "2"
StyleSuperSub 0
TypeText ".  The chemical formula for water is H"
StyleSuperSub 2
TypeText "2"
StyleSuperSub 0
TypeText "O."
```

Enters the following text:

Einstein's theory of relativity is $e=mc^2$. The chemical formula for water is H_2O.

See Also

StyleApplyStyle, StyleApplyFont, StylePointSize, StylePlain, StyleBold, StyleItalic, StyleUnderline, StyleStrikeout, StyleTextColor, GetStyleFont, DefineStyleFont

StyleLowercase

Description

Changes the selected text to lowercase. This command executes only when text is selected first.

Syntax

StyleLowercase

Example

```
Word 3, 1
StyleLowercase
```

Changes the three selected words to lowercase.

See Also

StyleUppercase, StyleTitlecase

StyleUppercase

Description

Changes the selected text to uppercase. This command executes only when text is selected first.

Syntax

StyleUppercase

Example

```
Word 3, 1
StyleUppercase
```

Changes the three selected words to uppercase.

See Also

StyleLowercase, StyleTitlecase

StyleTitlecase

Description

Changes the selected text to title case (initial capital letters). This command executes only when text is selected first.

Syntax

StyleTitlecase

Example

```
Word 3, 1
StyleTitlecase
```

Changes the three selected words to title case.

See Also

StyleLowercase, StyleUppercase

StyleTextColor

Description

Changes the color for selected text or for new text entered at the insertion point. To change the color setting in a paragraph style, use the DefineStyleColor statement.

Syntax

StyleTextColor

Parameters

colorValue% - The color to apply to text, a 24-bit color value.

Example

```
colorReturn% = RGB(255, 255, 255) + 1 ' color return to style
StyleTextColor RGB(255, 0, 0)    ' Red text
TypeText "This text is red.  "
StyleTextColor colorReturn%
TypeText "This text is in the original color."
```

Changes the text to red, then returns the color back to the original text color defined in the paragraph style.

See Also

RGB, HSI, CMY, StyleApplyStyle, StyleApplyFont, StylePointSize, StylePlain, StyleBold, StyleItalic, StyleUnderline, StyleStrikeout, StyleSuperSub, GetStyleColor, DefineStyleColor

StyleAlign

Description

Changes the alignment of the current or selected paragraphs.

Syntax

StyleAlign alignment%

Parameters

alignment% - The alignment to apply: 0 = left, 1 = center, 2 = right, 3 = justify.

Example

```
StyleAlign 1
TypeText "This text is centered.<Cr>"
StyleAlign 0
TypeText "This text is left-aligned.<Cr>"
```

See Also

StyleApplyStyle, DefineStyleParagraph

StyleModifyStyle

Description

Displays the Modify Paragraph Style dialog box.

Syntax

StyleModifyStyle

Example

```
StyleApplyStyle "Heading 1"
StyleModifyStyle
```

Applies the style Heading 1 to the current paragraph and displays the Modify Paragraph Style dialog box, allowing the user to modify the style.

See Also

DefineStyle, StyleApplyStyle

StyleCreateStyle

Description

Creates a new paragraph style based on an existing style.

Syntax

StyleCreateStyle styleName$, basedOn$, useCurrent%

Parameters

styleName$ - The name of the new style. The style name must not already exist in the document.

basedOn$ - The name of the style to copy from. This style must exist in the document. The default is the current paragraph style.

useCurrent% - If TRUE, use the local settings in the current paragraph for the new style. If FALSE, use only the settings in the original global style. The default is TRUE.

If no parameters are specified, the Create Paragraph Style dialog box appears.

Example

```
GetStyle styleName$, local%
StyleBold 1
StyleItalic 1
This text is bold and italic.
StyleCreateStyle "BoldItalic", styleName$, 1
StyleApplyStyle "BoldItalic"
```

Obtains the current style name, changes the text to bold and italic, and creates and applies a new style using those attributes.

See Also

StyleDeleteStyle

StyleDeleteStyle

Description

Deletes a paragraph style from the document.

Syntax

StyleDeleteStyle styleName$, current%

Parameters

styleName$ - The name of the style to delete.

current% - If TRUE and no style name is specified, deletes the current paragraph style. The default is FALSE.

If no parameters are specified, the Delete Paragraph Style dialog box appears.

Example

```
StyleDeleteStyle "Heading 5",
GetStyle styleName$, local%
IF styleName$ = "Heading 4" THEN StyleDeleteStyle , 1
```

Deletes the paragraph style Heading 5 and deletes the current paragraph style if it is Heading 4.

Note The BodyText paragraph style cannot be deleted.

See Also

StyleCreateStyle

StyleCreateLine

Description

Creates a new line style for use in tables, paragraph styles, and frame styles.

Syntax

ret% = StyleCreateLine (outsideSpace%, outsideThickness%, insideSpace%, insideThickness%)

Parameters

outsideSpace% - The space outside the lines, in decipoints.

outsideThickness% - The thickness of the outer line, in decipoints.

insideSpace% - The space between the two lines, in decipoints.

insideThickness% - The thickness of the inner line, in decipoints.

If no parameters are specified, the Create Line Style dialog box appears.

Returns

The number of the new line style in the array of line styles. The new line style entry is also updated in the WSW.INI file.

Example

```
ret% = StyleCreateLine(20, 10, 20, 50)
```

Creates a new double line style with 2 points space outside, a 1 point outer line, a 5 point inner line and 2 points of space between.

Note The new line style will appear in the line style dropdown boxes in the Modify Paragraph Style, Modify Frame Style, and Table Style dialog boxes. However, to define a style in a macro that uses the line style, you must specify the parameters individually in the DefineStyleLines statement.

See Also

StyleDeleteLine, DefineStyleLines

StyleDeleteLine

Description

Deletes a line style created with StyleCreateLine.

Syntax

StyleDeleteLine num%

Parameters

num% - The number returned by the StyleCreateLine function.

If no parameter is specified, the Delete Line Style dialog box appears.

Example

```
ret% = StyleCreateLine(20, 10, 20, 50)
StyleDeleteLine ret%
```

After creating the new line style, it is immediately deleted.

Note The line style is deleted from the WSW.INI file. The line style remains in effect in any paragraph or frame styles that have applied it, although it will no longer appear in the lists in the dialog boxes.

See Also

StyleCreateLine

StyleKeys

Description

Assigns a paragraph style or frame style to a function key.

Syntax

StyleKeys type%, styleName$, keyValue%

Parameters

type% - The type of style: 0 = paragraph style, 1 = frame style. The default is the first matching paragraph style, if found; otherwise, the first matching frame style.

styleName$ - The name of the style to assign.

keyValue% - The key to assign to the style. The value is the sum of the function key, from 2 to 9 or 11 to 16, plus 128 for the Shift key and 256 for the Ctrl key. The F1 and F10 keys cannot be assigned to styles.

Example

```
StyleKeys 0, "Heading 1", 2
StyleKeys 0, "Outline 1", 2 + 128
StyleKeys 1, "Page Border", 2 + 256
StyleKeys 1, "Page One", 2 + 128 + 256
```

Assigns the paragraph styles Heading 1 to F2 and Outline 1 to Shift+F2. Assigns the frame styles Page Border to Ctrl+F2 and Page One to Ctrl+Shift+F2.

StyleAutoNumModify

Description

Modifies the specified auto number style with new settings.

Syntax

StyleAutoNumModify autoNum$, level%, numberBy%, numberType%, digits%, textBefore$, combinePrevLevel%, delimiter$, textAfter$, attributes%, position%

Parameters

autoNum$ - The name of the auto number style to modify. The style must already exist.

level% - If autoNum$ is "Outline", the outline level to modify.

numberBy% - If 0, number by story; if 1, number by page.

numberType% - The numbering style: 1 = numeral, 2 = uppercase Roman, 3 = lowercase Roman, 4 = uppercase alpha, 5 = lowercase alpha.

digits% - If numberType% is 1, the number of digits to display.

textBefore$ - The text to display before the auto number.

combinePrevLevel% - If TRUE, autoNum$ is "Outline", and level% is greater than 1, combines this level with the previous level.

delimiter$ - If combinePrevLevel% is TRUE, the delimiter character to use between levels.

textAfter$ - The text to display after the auto number.

attributes% - The attributes to apply to the auto number: 0 = plain, 1 = bold, 2 = italic, 4 = underline. The values can be added together.

position% - The position of the auto number: 0 = at baseline, 1 = superscript, 2 = subscript.

The default for each parameter, except the number style name and level, is the current setting for the number style. If no parameters are specified, the Modify Auto Number Style dialog box appears.

Example

```
StyleAutoNumModify "Outline", 1, 0, 2, 1, "", 0, "", ".", 0, 0
StyleAutoNumModify "Outline", 2, 0, 4, 1, "", 0, "", ".", 0, 0
StyleAutoNumModify "Outline", 3, 0, 1, 1, "", 0, "", ".", 0, 0
StyleAutoNumModify "Outline", 4, 0, 5, 1, "", 0, "", ")", 2, 0
StyleAutoNumModify "Outline", 5, 0, 1, 1, "(", 0, "", ")", 0, 0
StyleAutoNumModify "Outline", 6, 0, 5, 1, "(", 0, "", ")", 2, 0
StyleAutoNumModify "Outline", 7, 0, 3, 1, "", 0, "", ")", 0, 0
StyleAutoNumModify "Outline", 8, 0, 5, 1, "", 0, "", ".", 2, 0
```

Defines the Outline numbering style for the standard outline style: I., A., 1., a., etc.

See Also

StyleAutoNumCreate, StyleAutoNumDelete, DefineStyleNumbering

StyleAutoNumCreate

Description

Creates a new auto number style from an existing one.

Syntax

StyleAutoNumCreate autoNum$, basedOn$

Parameters

autoNum$ - The name of the new auto number style. The name must not already exist in the document.

basedOn$ - The name of an existing auto number style to copy for the new style.

If no parameters are specified, the Auto Number Style dialog box appears.

Example

```
StyleAutoNumCreate "RomanPage", "Page"
StyleAutoNumModify "RomanPage",,, 3,,,,,,,
```

Creates a new auto number style, called RomanPage, that uses lowercase Roman numerals. The remaining settings are the same as the original Page number style.

Note A new style created from the Outline number style will use the settings from level 1 of the Outline number style.

See Also

StyleAutoNumModify, StyleAutoNumDelete

StyleAutoNumDelete

Description

Deletes an auto number style.

Syntax

StyleAutoNumDelete autoNum$

Parameters

autoNum$ - The name of the auto number style to delete.

If no parameters are specified, the Auto Number Style dialog box appears.

Example

```
StyleAutoNumDelete "RomanPage"
```

Deletes the RomanPage auto number style.

Note The Outline, Footnote, and Page auto number styles cannot be deleted.

See Also

StyleAutoNumCreate, StyleAutoNumModify

StyleFootnote

Description

Modifies the footnote settings for the current document.

Syntax

StyleFootnote styleName$, spaceAbove%, maxArea%, thickness%, width%, alignment%, toText$, fromText$

Parameters

styleName$ - The default paragraph style to use for new footnotes. Any existing footnotes will retain their original paragraph styles.

spaceAbove% - The amount of space between the text on the page and the first footnote, in decipoints.

maxArea% - The maximum percentage of the page to allow for footnotes, from 10 to 100 percent.

thickness% - The thickness of the line separating the text and footnotes, in decipoints. If 0, no separator line appears.

width% - The width of the separator line, in decipoints. If set to -1, makes the separator line the width of the page.

alignment% - The alignment of the separator line: 0 = left, 1 = centered.

toText$ - The text to display at the end of a footnote that continues to the next page. You can use WSWin text conventions.

fromText$ - The text to display at the beginning of a continued footnote. You can use WSWin text conventions.

The default for the each setting is to leave it unchanged. If no parameters are specified, the Footnote Style dialog box appears.

Example

```
StyleFootnote "Footnote", To_DP(".25", 0), 80, 10, 1440, 1,,
```

Changes the footnote settings to use the Footnote paragraph style, one-quarter inch above the footnotes, 80% maximum footnote area, and a one-point separator line that is two inches long and centered.

See Also

DefineStyle

DefineStyle

Description

Begins a paragraph style definition segment. Once a style definition begins, no other macro statements except for style definition statements can be executed. Use EndDefineStyle to complete a style definition segment.

Syntax

DefineStyle styleName$, local%

Parameters

styleName$ - The name of the style to modify. The style must already exist in the document.

local% - If TRUE, changes the style for the current paragraph only. If FALSE, changes the global paragraph style. The default is FALSE.

Example

```
tab1% = To_DP("3.25", 0)
tab2% = To_DP("6.5", 0)
textColor% = RGB(0, 0, 0)
lineColor% = RGB(255, 0, 0)
effectColor% = RGB(0, 0, 255)
StyleCreateStyle "NewStyle", "BodyText",
DefineStyle "NewStyle", 0
DefineStyleNextStyle "NewStyle"
DefineStyleFont "Arial", 100, 120,,,,,,,
DefineStyleParagraph 50, 0, 0, 0, 0, 3,,,, 3,,, 1
DefineStyleKerning 1, 90,,
DefineStyleTabClearAll
DefineStyleTabSet 1, 0, tab1%
DefineStyleTabSet 2, 0, tab2%
DefineStyleColor textColor%, lineColor%, effectColor%
DefineStyleLines 1, -1, 0, 2,,,,
DefineStyleSuperSub 25, 0, 0
DefineStyleWordSpacing 50, 120, 1
EndDefineStyle 1
```

Creates a new style called NewStyle based on the BodyText style. Changes NewStyle to the following settings:

- The font is Arial 10 point with a 12-point line height.
- The paragraph has a one-half inch first indent, is justified, allows three consecutive hyphens, and controls widows and orphans.
- The paragraph does pair kerning above 9 points.
- Tab stops are set at 3.25 inches and 6.5 inches.
- The text is black, with red lines and blue effects.
- A line appears below the text, the width of the column, using the second predefined line style (2-point single line).
- Superscript/subscript text is reduced 25 percent of normal size, underlined below descenders without increasing line height.
- The paragraph allows a minimum 50 percent and maximum 120 percent word spacing with letter spacing on.

Finally, the macro applies the NewStyle paragraph style to the current paragraph.

Note You cannot use functions within frame style definitions to change settings. As shown in the example above, assign the values to variables before you start the frame style definition.

See Also

DefineStyle, DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleNextStyle

Description

Specifies the next paragraph style setting for the paragraph style.

Syntax

DefineStyleNextStyle styleName$

Parameters

styleName$ - The name of the style to apply to the next paragraph. The default is no change to the paragraph style.

Example

See DefineStyle.

See Also

DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleFont

Description

Specifies the font attributes for the paragraph style.

Syntax

DefineStyleFont typeFace$, size%, lineHeight%, bold%, italic%, underline%, strikeout%, hidden%, placeUnderline%, strikeoutChar$

Parameters

typeFace$ - The name of the font to use.

size% - The size of the font, in decipoints.

lineHeight% - The line height for the style, in decipoints.

bold% - The bold attribute: 0 = off, 1 = on.

italic% - The italic attribute: 0 = off, 1 = on.

underline% - The underline attribute: 0 = off, 1 = single, 2 = double.

strikeout% - The strikeout attribute: 0 = off, 1 = on.

hidden% - The hidden attribute: 0 = off, 1 = on.

placeUnderline% - The location for the underline: 0 = at baseline, 1 = below descenders.

strikeoutChar$ - The character to use for strikeout.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleParagraph

Description

The paragraph settings for the paragraph style.

Syntax

DefineStyleParagraph firstIndent%, leftIndent%, rightIndent%, spaceAbove%, spaceBelow%, alignment%, location%, spanColumns%, allowHyphenBreak%, numHyphen%, keepWithNext%, allowBreakWithin%, widowControl%

Parameters

firstIndent% - The first indent setting, in decipoints.

leftIndent% - The left indent setting, in decipoints.

rightIndent% - The right indent setting, in decipoints.

spaceAbove% - The space above setting, in decipoints.

spaceBelow% - The space below setting, in decipoints.

alignment% - The alignment setting: 0 = left, 1 = center, 2 = right, 3 = justified.

location% - The location of the paragraph: 0 = normal, 1 = top of frame or page, 2 = top of column.

spanColumns% - The span columns setting: 0 = off, 1 = on.

allowHyphenBreak% - The auto hyphenation setting: 0 = off, 1 = on.

numHyphen% - The number of consecutive hyphens to allow, from 1 to 9.

keepWithNext% - The keep with next paragraph setting: 0 = off, 1 = on.

allowBreakWithin% - The allow break within paragraph setting: 0 = off, 1 = on.

widowControl% - The widow control setting: 0 = off, 1 = on.

The default for each setting is to leave the setting in the paragraph style unchanged.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleKerning

Description

The kerning settings for the paragraph style.

Syntax

DefineStyleKerning autoPairKern%, aboveSize%, trackKern%, trackAmount%

Parameters

autoPairKern% - The automatic pair kerning setting: 0 = off, 1 = on.

aboveSize% - The font size above which pair kerning will take effect, in decipoints.

trackKern% - The track kerning setting: 0 = off, 1 = on.

trackAmount% - The amount of track kerning to apply, in 1/1000s of an em, from 0 to 2000.

The default for each setting is to leave the setting in the paragraph style unchanged.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleTabSet

Description

Defines a tab stop in a paragraph style. You can also use the statement to modify an existing tab stop alignment or leader character.

Syntax

DefineStyleTabSet type%, leader%, location%

Parameters

type% - The type of tab to set: 0 = left, 1 = center, 2 = right, 3 = decimal.

leader% - The type of tab leader to use: 0 = none, 1 = dot leader, 2 = line leader.

location% - The location of the tab stop, in decipoints.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleTabClear

Description

Clears a tab stop from a paragraph style.

Syntax

DefineStyleTabClear location%

Parameters

location% - The position of the tab stop to clear, in decipoints.

Example

```
DefineStyleTabClear To_DP("3.25")
```

Clears the tab stop at 3.25 inches.

See Also

DefineStyle, DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleTabClearAll

Description

Clears all tab stops from a paragraph style.

Syntax

DefineStyleTabClearAll

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleColor

Description

Defines the color of text, lines, and special effects in a paragraph style.

Syntax

DefineStyleColor textColor%, lineColor%, effectColor%

Parameters

textColor% - The color for text in the style, as a 24-bit color value.

lineColor% - The color for lines and boxes, as a 24-bit color value.

effectColor% - The color for bullet or drop caps, as a 24-bit color value.

The default for each setting is to leave the setting in the paragraph style unchanged.

Example

See DefineStyle.

Note Before assigning the color for a special effect, you should define the special effect.

See Also

RGB, HSI, CMY, DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleLines

Description

Specifies the lines or box for a paragraph style.

Syntax

DefineStyleLines position%, length%, alignment%, lineStyle%, outsideSpace%, outsideThickness%, spaceBetween%, insideThickness%

Parameters

position% - The position of the lines: 0 = line above, 1 = line below, 2 = box around.

length% - The length of the line, in decipoints. If 0, no line will appear in the specified position%; if -1, the line will span the entire column; if -2, the line will span between the left and right indent settings.

alignment% - The alignment of the line: 0 = left, 1 = center, 2 = right.

lineStyle% - The line style to use: 0 = none, 1 = one point single, 2 = two point single, 3 = four-point single, 4 = one-point double, 5 = two-point double, 6 = four-point double, 7 = other.

outsideSpace% - If lineStyle% = 7, the space outside the lines, in decipoints.

outsideThickness% - If lineStyle% = 7, the thickness of the outer line, in decipoints.

spaceBetween% - If lineStyle% = 7, the space between the lines, in decipoints.

insideThickness% - If lineStyle% = 7, the thickness of the inner line, in decipoints.

Note If position% is 2, double line settings are ignored.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleBullet

Description

Defines the bullet settings for a paragraph style.

Syntax

DefineStyleBullet type%, filled%, position%

Parameters

type% - The type of bullet to use: 0 = none, 1 = round, 2 = square, 3 = hyphen.

filled% - Whether the bullet should be open or filled: 0 = open, 1 = filled. This parameter is ignored if type% = 3.

position% - The position of the bullet, in decipoints. It must be to the left of the first indent.

The default for each setting is to leave the setting in the paragraph style unchanged.

Example

```
StyleCreateStyle "Bullet", "BodyText",
DefineStyle "Bullet"
DefineStyleBullet 1, 1, 0
DefineStyleParagraph 180, 180,,,,,,,,,,
EndDefineStyle
```

Creates a new style called Bullet and assigns a round filled bullet to the new style. The first and left indents are changed to one-quarter inch.

See Also

DefineStyle, DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleNumbering

Description

Defines the numbering settings for a paragraph style.

Syntax

DefineStyleNumbering numberStyle$, level%, position%, alignment%

Parameters

numberStyle$ - The name of an auto number style. The auto number style must exist in the document.

level% - If the Outline number style is used, the level of the number style to use, from 1 to 8.

position% - The position of the number in the paragraph style, in decipoints. It must be to the left of the first indent.

alignment% - The alignment of the number: 0 = left, 2 = right. This parameter is ignored if position% = 0.

The default for each setting is to leave the setting in the paragraph style unchanged.

Example

```
StyleCreateStyle "Number", "BodyText",
StyleAutoNumCreate "NumStyle", "Page"
StyleAutoNumModify "NumStyle",, 0, 1, 1,,,, ".",,
DefineStyle "Number"
DefineStyleNumbering "NumStyle",, 0, 0
DefineStyleParagraph 180, 180,,,,,,,,,,
EndDefineStyle
```

Creates a new style called Number and a new auto number style called NumStyle. The new auto number style is numeric with a period after the number. The Number paragraph style is modified to include the NumStyle auto number style, left aligned at the left margin. The first and left indents are changed to one-quarter inch to make room for the auto number.

See Also

DefineStyle, DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleDropCap

Description

Defines the drop cap settings for a paragraph style.

Syntax

DefineStyleDropCap typeFace$, size%, lineHeight%, bold%, italic%, alignBaseline%, numCharacters%, adjustBaseline%

Parameters

typeFace$ - The font to use for the drop cap. The default is the same as the paragraph style font.

size% - The point size for the drop cap, in decipoints. The default is double the font size in the paragraph style.

lineHeight% - The line height for the drop cap, in decipoints. The default is 73 percent of the drop cap point size.

bold% - The bold attribute for the drop cap: 0 = off, 1 = on.

italic% - The italic attribute for the drop cap: 0 = off, 1 = on.

alignBaseline% - The alignment setting for the drop cap: 0 = align with top of characters, 1 = align with nearest baseline.

numCharacters% - The number of characters for the drop cap: from 1 to 9. If 0 is specified, then the drop cap setting is turned off for the paragraph style.

adjustBaseline% - The additional baseline adjust setting, in decipoints.

Example

```
StyleCreateStyle "DropCap", "BodyText",
DefineStyle "DropCap"
DefineStyleDropCap , 36,, 1, 0, 0, 1, 0
EndDefineStyle
```

Creates a new style called DropCap and assigns a drop cap to the style that is 36 points tall and bold.

See Also

DefineStyle, DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

DefineStyleSuperSub

Description

Specifies the superscript and subscript settings for a paragraph style.

Syntax

DefineStyleSuperSub reduceBy%, underlineAtBaseline%, increaseHeight%

Parameters

reduceBy% - The percentage reduction to apply to superscript and subscript characters, from 0 to 75. For example, a value of 25 would reduce superscript and subscript characters to 75 percent of their original size.

underlineAtBaseline% - If TRUE, the underline adjusts to follow the baseline of the superscript or subscript.

increaseHeight% - If TRUE, the line height increases when a superscript or subscript character is on the line.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleWordSpacing, EndDefineStyle

DefineStyleWordSpacing

Description

Specifies the word spacing settings for a paragraph style. These settings are used when text is justified.

Syntax

DefineStyleWordSpacing minWordSpacing%, maxWordSpacing%, letterSpacing%

Parameters

minWordSpacing% - The minimum word spacing percentage to apply, from 50 to 100. A value of 50 specifies that the word spacing can be reduced to half if needed to justify text. A value of 100 specifies that the word spacing cannot be smaller than normal.

maxWordSpacing% - The maximum word spacing percentage to apply, from 100 to 500. A value of 100 specifies that word spacing cannot be larger than normal. A value of 500 specifies that word spacing cannot be adjusted up to five times the normal word spacing to justify text.

letterSpacing% - If TRUE, then the spaces between letters can be adjusted to justify text.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

EndDefineStyle

Description

Ends a paragraph style definition segment.

Syntax

EndDefineStyle apply%

Parameters

apply% - If TRUE, the style that was modified in the segment is applied to the current paragraph or selected text. The default is TRUE if the local% parameter at the beginning of the style definition was TRUE, and FALSE if the local% parameter was FALSE.

Example

See DefineStyle.

See Also

DefineStyleNextStyle, DefineStyleFont, DefineStyleParagraph, DefineStyleKerning, DefineStyleTabSet, DefineStyleTabClear, DefineStyleTabClearAll, DefineStyleColor, DefineStyleLines, DefineStyleBullet, DefineStyleNumbering, DefineStyleDropCap, DefineStyleSuperSub, DefineStyleWordSpacing, EndDefineStyle

Frame Statements

The Frame Statements section describes the available commands on the WSWin Frame menu. The Frame Statements include

- FrameApplyFrameStyle
- FrameBringFront
- FrameSendBack
- FrameLinkFrame
- FrameUnlinkFrame
- FrameSnapTo
- FrameFloatFixed
- FrameModifyFrameStyle
- FrameCreateFrameStyle
- FrameDeleteFrameStyle
- FrameSize
- FramePageOrientation
- DefineFrameStyle
- DefineFrameStyleDefaultStyle
- DefineFrameStyleMargins
- DefineFrameStyleOptions
- DefineFrameStyleBorders
- DefineFrameStyleColor
- DefineFrameStyleColumns
- DefineFrameStyleGutterLines
- DefineFrameStyleScaling
- DefineFrameStyleTable
- EndDefineFrameStyle

FrameApplyFrameStyle

Description

Applies a frame style to the current frame.

Syntax

FrameApplyFrameStyle styleName$

Parameters

styleName$ - The name of the frame style to apply. The style must already exist in the document, and it must be the correct frame style type for the frame (for example, only text frame styles can apply to text frames).

Example

```
FrameApplyFrameStyle "Text"
```

Applies the frame style Text to the current frame.

See Also

FrameModifyFrameStyle, FrameCreateFrameStyle, FrameDeleteFrameStyle, DefineFrameStyle

FrameBringFront

Description

Brings the current frame in front of its sibling frames.

Syntax

FrameBringFront

Example

```
NextFrame
FrameBringFront
```

Goes to the next frame and brings it to the front.

See Also

FrameSendBack

FrameSendBack

Description

Sends the current frame behind its sibling frames.

Syntax

FrameSendBack

Example

```
PrevFrame
FrameSendBack
```

Goes to the previous frame and sends it to the back.

See Also

FrameBringFront

FrameLinkFrame

Description

Links a frame to another frame of the same type.

Syntax

FrameLinkFrame toFrame%

Parameters

toFrame% - The frame number of another frame in the document. The frame must be of the same type as the current frame. The frame cannot be a page frame if Auto Pagination is turned on.

Example

```
frame1% = CreateFrame(0, 720, 720, 2160, 3600, "Text", )
ParentFrame
frame2% = CreateFrame(0, 2160, 720, 2160, 3600, "Text", )
EditGoto 9, frame1%,,
FrameLinkFrame frame2%
```

Creates two frames on the page frame, then goes to the first frame and links it to the second frame.

See Also

FrameUnlinkFrame

FrameUnlinkFrame

Description

Unlinks the current frame from its chain.

Syntax

FrameUnlinkFrame

Example

```
NextFrame
UnlinkFrame
```

Goes to the next frame and unlinks it.

Note If the current frame is a page frame and Auto Pagination is on, the frame cannot be unlinked.

See Also

FrameLinkFrame

FrameSnapTo

Description

Specifies the snap-to grid settings for the document.

Syntax

FrameSnapTo snapTo%, display%, originX%, originY%, horzGrid%, vertGrid%

Parameters

snapTo% - The snap-to setting: 0 = off, 1 = snap to frames, 2 = snap to margins, 4 = snap to page grid. Add numbers together for multiple settings.

display% - The display settings: 0 = off, 8 = display margins, 16 = display page grid. Add numbers together for multiple settings.

originX% - The horizontal origin of the page grid, in decipoints. The default is 0.

originY% - The vertical origin of the page grid, in decipoints. The default is 0.

horzGrid% - The horizontal grid spacing, in decipoints. The default is 90 (one-eighth inch).

vertGrid% - The vertical grid spacing, in decipoints. The default is 90 (one-eighth inch).

Example

```
FrameSnapTo 4, 0, 0, 0, 180, 180
```

Turns snap on for the page grid only; no grid or margins display when frames are moved. The origin of the grid is 0, 0; and the grid spacing is one-quarter inch square.

See Also

ViewDisplay

FrameFloatFixed

Description

Toggles the floating/fixed attribute of the current frame.

Syntax

FrameFloatFixed

Example

```
NextFrame
float% = GetFrameStatus()
IF float% THEN FrameFloatFixed
```

Goes to the next frame and gets the floating status of the frame. If it is floating, the status of the frame is changed to fixed.

See Also

GetFrameStatus

FrameModifyFrameStyle

Description

Displays the Modify Frame Style dialog box.

Syntax

ModifyFrameStyle

Example

```
FrameApplyFrameStyle "Text"
FrameModifyFrameStyle
```

Applies the frame style Text to the current frame and displays the Modify Frame Style dialog box for the current frame style.

See Also

DefineFrameStyle

FrameCreateFrameStyle

Description

Creates a new frame style based on an existing frame style.

Syntax

FrameCreateFrameStyle styleName$, basedOn$, useCurrent%

Parameters

styleName$ - The name of the new frame style. The style must not already exist in the document.

basedOn$ - The frame style to base the new style on. The style must already exist in the document. The default is the current frame style.

useCurrent% - If TRUE, define the new style using any local attributes in the current frame. If FALSE, use the global frame style as the base. The default is TRUE if the frame has local settings.

Example

```
FrameCreateFrameStyle "TextBorder", "Text", 0
```

Creates a new frame style called TextBorder, based on the Text frame style.

See Also

FrameDeleteFrameStyle, DefineFrameStyle

FrameDeleteFrameStyle

Description

Deletes a frame style from the document.

Syntax

FrameDeleteFrameStyle styleName$, current%

Parameters

styleName$ - The name of the style to delete. The frame styles PageText, Text, Caption, EPS, Graphic, HeadFoot, Object, Table, and Text cannot be deleted.

current% - If styleName$ is not specified and current% is TRUE, delete the frame style for the current frame.

Example

```
FrameDeleteFrameStyle "TextBorder",
```

Deletes the frame style TextBorder from the document. Any frames using that frame style will revert to the default frame styles for the frame type.

See Also

FrameCreateFrameStyle, DefineFrameStyle

FrameSize

Description

Specifies the position and size of the current frame.

Syntax

FrameSize originX%, originY%, width%, height%

Parameters

originX% - The horizontal position of the frame, relative to the left edge of the parent frame, in decipoints.

originY% - The vertical position of the frame, relative to the top edge of the parent frame, in decipoints.

width% - The width of the frame, in decipoints.

height% - The height of the frame, in decipoints.

Example

```
FrameSize 720, 720, 1440, 2880
```

Resizes the frame to two inches wide by four inches tall, positioned one inch from the top and left edge of the parent frame.

See Also

InsertFrame

FramePageOrientation

Description

Changes the orientation of the current page.

Syntax

FramePageOrientation orientation%

Parameters

orientation% - The orientation of the page: 0 = protrait, 1 = landscape, 2 = toggle. The default is 2.

Example

```
FramePageOrientation 1
```

Changes the current page to landscape.

See Also

GetTextFrameType

DefineFrameStyle

Description

Begins a frame style definition segment. Once a frame style definition begins, no other macro statements except for frame style definition statements can be executed. Use EndDefineFrameStyle to complete a frame style definition segment.

Syntax

DefineFrameStyle styleName$, local

Parameters

styleName$ - The name of the frame style to modify. The style must already exist in the document.

local% - If TRUE, changes the frame style for the current frame only. If FALSE, changes the global frame style. The default is FALSE.

Example

```
backColor% = RGB(255, 255, 0)
borderColor% = RGB(128, 0, 128)
gutterColor% = RGB(0, 128, 128)
FrameCreateFrameStyle "NewText", "Text",
DefineFrameStyle "NewText", 0
DefineFrameStyleDefaultStyle "BodyText"
DefineFrameStyleMargins 180, 180, 180, 180
DefineFrameStyleOptions 0, 1, 0
DefineFrameStyleBorders 1, 1, 0,,,,
DefineFrameStyleBorders 2, 3, 0,,,,
DefineFrameStyleBorders 3, 1, 0,,,,
DefineFrameStyleBorders 4, 3, 0,,,,
DefineFrameStyleColor backColor%, borderColor%, gutterColor%
DefineFrameStyleColumns 2, 180, 0, 1, 1440
DefineFrameStyleGutterLines 1, 0, 7, 10, 10, 10
EndDefineFrameStyle 1
```

Creates a new frame style called NewText based on the Text frame style. Modifies the NewText frame style as follows:

- The default paragraph style is BodyText.
- The margins are one-quarter inch all around.
- The frame is not transparent, text wraps around the frame, the background fills to the frame edge.
- The frame has one-point borders top and left and four-point borders right and bottom.
- The frame background color is yellow, the frame border color is dark magenta, and the gutter line color is dark cyan.
- The frame has two columns that are uneven with the first column two inches wide.
- The frame has a double one-point gutter line the height of the margins with one point of space between the lines.

Note You cannot use functions within frame style definitions to change settings. As shown in the example above, assign the values to variables before you start the frame style definition.

See Also

DefineFrameStyle, DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleDefaultStyle

Description

Specifies the default paragraph style for a frame style. This setting only applies to text frame styles.

Syntax

DefineFrameStyleThe defaultStyle styleName$

Parameters

styleName$ - The name of a paragraph style to apply to text by default.

Example

See DefineFrameStyle.

See Also

DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleMargins

Description

Specifies the margins for a frame style.

Syntax

DefineFrameStyleMargins left%, right%, top%, bottom%

Parameters

left% - The left margin, in decipoints.

right% - The right margin, in decipoints.

top% - The top margin, in decipoints.

bottom% - The bottom margin, in decipoints.

The default for each setting is to leave the setting in the frame style unchanged.

Example

See DefineFrameStyle.

See Also

DefineFrameStyleDefaultStyle, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleOptions

Description

Specifies other frame style options in a frame style.

Syntax

DefineFrameStyleOptions transparent%, textWrapAround%, backgroundFill%

Parameters

transparent% - The transparency setting for the frame style: 0 = not transparent, 1 = transparent.

textWrapAround% - The text wrap setting: 0 = text doesn't wrap around, 1 = text wraps around.

backgroundFill% - The setting for the background color fill: 0 = fill to edge of frame, 1 = fill to outer border, 2 = fill to inner border.

Example

See DefineFrameStyle.

See Also

DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleBorders

Description

Specifies the frame border settings for a frame style.

Syntax

DefineFrameStyleBorders position%, lineStyle%, radius%, outsideSpace%, outsideThickness%, spaceBetween%, insideThickness%

Parameters

position% - The position of the border: 0 = none, 1 = left (inside for facing pages), 2 = right (outside for facing pages), 3 = top, 4 = bottom, 5 = all sides.

lineStyle% - The line style for the border position: 0 = none, 1 = one-point single, 2 = two-point single, 3 = four-point single, 4 = one-point double, 5 = two-point double, 6 = four-point double, 7 = other.

radius% - If position% = 5, the radius of the border if rounded corners are desired, in decipoints. Set radius% = 0 for no rounded counters. (The value adjusts automatically if it is too large for the selected line style and margin settings.)

outsideSpace% - If lineStyle% = 7, the space outside the lines, in decipoints.

outsideThickness% - If lineStyle% = 7, the thickness of the outer line, in decipoints.

spaceBetween% - If lineStyle% = 7, the space between the lines, in decipoints.

insideThickness% - If lineStyle% = 7, the thickness of the inner line, in decipoints.

Example

See DefineFrameStyle.

See Also

DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleColor

Description

Specifies the color settings for a frame style.

Syntax

DefineFrameStyleColor backgroundColor%, borderColor%, gutterLineColor%

Parameters

backgroundColor% - The color of the frame background as a 24-bit color value.

borderColor% - The color of the frame border as a 24-bit color value.

gutterLineColor% - The color of the gutter lines as a 24-bit color value (for text frames only).

Example

See DefineFrameStyle.

See Also

RGB, HSI, CMY, DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleColumns

Description

Specifies the column settings for a frame style. This setting is valid only for text frame styles.

Syntax

DefineFrameStyleColumns numcols%, gutterWidth%, balanced%, unevenCols%, firstColWidth%

Parameters

numcols% - The number of columns from 1 to 32.

gutterWidth% - The width of the space between columns, in decipoints.

balanced% - The balanced column setting: 0 = off, 1 = on.

unevenCols% - Uneven columns: 0 = off, 1 = on. This is valid only if the number of columns is 2.

firstColWidth% - If uneven columns is on, the width of the first column, in decipoints.

Example

See DefineFrameStyle.

See Also

DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleGutterLines

Description

Specifies the gutter line settings for a frame style. This setting is valid only for text frame styles.

Syntax

DefineFrameStyleGutterLines height%, inset%, lineStyle%, leftLine%, spaceBetween%, rightLine%

Parameters

height% - The height of the gutter lines: 0 = height of the frame, 1 = from margin to margin, 2 = the height of the text.

inset% - The additional amount to inset from the gutter line height, in decipoints.

lineStyle% - The line style to use for the gutter lines: 0 = none, 1 = one-point single, 2 = two-point single, 3 = four-point single, 4 = one-point double, 5 = two-point double, 6 = four-point double, 7 = other.

leftLine% - If lineStyle% = 7, the width of the left gutter line, in decipoints.

spaceBetween% - If lineStyle% = 7, the width of the space between the gutter lines, in decipoints.

rightLine% - If lineStyle% = 7, the width of the right gutter line, in decipoints.

Example

See DefineFrameStyle.

See Also

DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleScaling, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleScaling

Description

Specifies the scaling setting for a frame style. This setting is valid only for graphic, OLE, and EPS frame styles.

Syntax

DefineFrameStyleScaling scaling%, horzPos%, vertPos%

Parameters

scaling% - The scaling setting for the frame style: 0 = cropped, 1 = scale to fit frame, 2 = scale to fit and maintain aspect ratio.

horzPos% - If scaling% = 2, the horizontal position of the frame contents: 0 = left, 1 = center, 2 = right.

vertPos% - If scaling% = 2, the vertical position of the frame contents: 0 = top, 1 = center, 2 = bottom.

Example

```
DefineFrameStyle "Object",
DefineFrameStyleScaling 2, 1, 1
EndDefineFrameStyle
```

Modifies the Object frame style so that scaling is turned on with the aspect ratio maintained. If the frame contents cannot scale in both directions proportionately, the object is centered within the frame.

See Also

DefineFrameStyle, DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleTable, EndDefineFrameStyle

DefineFrameStyleTable

Description

Specifies the table frame settings for a frame style. This setting is valid only for table frame styles.

Syntax

DefineFrameStyleTable showFirstRow%, displayNextRow%, displayNextCol%

Parameters

showFirstRow% - If TRUE, the first row of the table always appears in the frame.

displayNextRow% - If TRUE, a linked frame displays the next row from the previous frame in the chain

displayNextCol% - If TRUE, a linked frame displays the next column from the previous frame in the chain

Example

```
DefineFrameStyle "Table",
DefineFrameStyleTable 1, 1, 0
EndDefineFrameStyle
```

Modifies the Table frame style so that the first row always appears, and linked frames display the next row but not the next column.

See Also

DefineFrameStyle, DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, EndDefineFrameStyle

EndDefineFrameStyle

Description

Ends a frame style definition segment.

Syntax

EndDefineFrameStyle apply%

Parameters

apply% - If TRUE, the style that was modified in the segment is applied to the current frame. The default is TRUE if the local% parameter at the beginning of the style definition was TRUE, and FALSE if the local% parameter was FALSE.

Example

See DefineFrameStyle.

See Also

DefineFrameStyleDefaultStyle, DefineFrameStyleMargins, DefineFrameStyleOptions, DefineFrameStyleBorders, DefineFrameStyleColor, DefineFrameStyleColumns, DefineFrameStyleGutterLines, DefineFrameStyleScaling, DefineFrameStyleTable

Table Statements

The Table Statements section describes the available commands on the WSWin Table menu. The Table Statements include

- TableFormatTable
- TableFormatRow
- TableFormatColumn
- TableLines
- TableColor
- TableSelectRow
- TableSelectColumn
- TableInsertRow
- TableInsertColumn
- TableDeleteRow
- TableDeleteColumn

Note The Table Statements are not recorded by WSWin; however, they are available when you want to write your own macros from scratch.

TableFormatTable

Description

Specifies the width, number of rows and columns, and fill direction for the current table. This command is available only when a table frame is active.

Syntax

TableFormatTable width%, numRows%, numCols%, fillBy%

Parameters

width% - The width of the table, in decipoints. The default is the width of the frame.

numRows% - The number of rows in the table. The default is 3.

numCols% - The number of columns in the table. The default is the width of the table in inches, ignoring any fraction.

fillBy% - The fill direction: 0 = fill by row; 1 = fill by column. The default is 0.

If no parameters are specified, the Format Table dialog box appears.

Example

```
TableFormatTable 2880, 3, 4, 0
```

Formats the current table to be four inches wide with three rows and four columns.

See Also

TableFormatRow, TableFormatColumn, TableLines, TableColor

TableFormatRow

Description

Specifies the margins and height of a specific row of a table. This command is available only when a table frame is active.

Syntax

TableFormatRow rowNum%, top%, bottom%, height%

Parameters

rowNum% - The row number to change. If not specified, the command affects all rows in the table except for the first row. The first row of the table must be formatted separately.

top% - The top margin for the row, in decipoints.

bottom% - The bottom margin for the row, in decipoints.

height% - The height of the row, in decipoints. To allow the row to adjust automatically so it fits the contents of the cells, specify 0. Any positive value forces the row height to that value. The default is 0.

The default for all other settings is to leave the setting in the table unchanged. If no parameters are specified, the Format Row dialog box appears.

Example

```
TableFormatRow 1, 40, 40, 0
TableFormatRow  , 20, 20, 0
```

Formats the first row of the table to have a four-point top and bottom margin. Formats the remaining rows of the table to have two-point top and bottom margins. All rows can expand to fit the text in the cells.

See Also

TableFormatColumn

TableFormatColumn

Description

Specifies the width, margins, and divisions of the columns in a table. This command is available only when a table frame is active.

Syntax

TableFormatColumn colNum$, desc$, width%, left%, right%, subCol%

Parameters

colNum$ - The number of the column to change. To specify a subcolumn, separate the column numbers with a period. For example: the first subcolumn of column two would be specified as "2.1". The parameter is a string, so it must be enclosed with double quotation marks. If not specified, the command affects all columns in the table.

desc$ - The description for the column.

width% - The width of the column, in decipoints.

left% - The left margin for the column, in decipoints.

right% - The right margin for the column, in decipoints.

subCol% - The number of subcolumns, if the column is to be split. If the column should not be split, specify 0. The default is 0.

The default for all other settings is to leave the setting in the table unchanged. If no parameters are specified, the Format Column dialog box appears.

Example

```
TableFormatColumn ,, To_DP(".75",0), 40, 40, 0
TableFormatColumn "1", "Column 1", To_DP("3",0),,,
```

Formats all columns in the table to be three-quarters of an inch wide, then reformats the first column to be three inches wide, with its own description.

See Also

TableFormatRow

TableLines

Description

Specifies the type of lines to use between the rows and columns of a table. This command is available only when a table frame is active.

Syntax

TableLines position%, lineStyle%, outsideSpace%, outsideThickness%, spaceBetween%, insideThickness%

Parameters

position% - The line position to change: 1 = left, 2 = right, 3 = top, 4 = bottom, 5 = all, 6 = horizontal, 7 = vertical. Values 1 through 5 affect the outside lines of the table, 6 and 7 affect the lines between rows and columns.

lineStyle% - The line style for the border position: 0 = none, 1 = one-point single, 2 = two-point single, 3 = four-point single, 4 = one-point double, 5 = two-point double, 6 = four-point double, 7 = other.

outsideSpace% - If lineStyle% = 7, the space outside the lines, in decipoints.

outsideThickness% - If lineStyle% = 7, the thickness of the outer line, in decipoints.

spaceBetween% - If lineStyle% = 7, the space between the lines, in decipoints.

insideThickness% - If lineStyle% = 7, the thickness of the inner line, in decipoints.

The default for all other settings is to leave the setting in the table unchanged. If no parameters are specified, the Lines section of the Table Style dialog box appears.

Example

```
TableLines 5, 2,,,,
TableLines 6, 7, 0, 5, 10, 5
TableLines 7, 7, 0, 5, 10, 5
```

Specifies a two-point single line around the outside of the table, and a half-point double line, separated by one point of space, between the rows and columns.

See Also

TableColor

TableColor

Description

Specifies the color settings for a table. This command is available only when a table frame is active.

Syntax

TableColor lineColor%, backgrndColor%

Parameters

lineColor% - The color of the table lines, as a 24-bit color value.

backgrndColor% - The color of the table background, as a 24-bit color value.

The default for all other settings is to leave the setting in the table unchanged. If no parameters are specified, the Color section of the Table Style dialog box appears.

Example

```
TableColor RGB(255, 0, 0), RGB(255, 255, 0)
```

Changes the current table to have red lines and a yellow background.

See Also

RGB, CMY, HSI

TableSelectRow

Description

Selects the table row containing the insertion point. This command is available only when a table frame is active.

Syntax

TableSelectRow

See Also

TableSelectColumn

TableSelectColumn

Description

Selects the table column containing the insertion point. This command is available only when a table frame is active.

Syntax

TableSelectColumn

See Also

TableSelectRow

TableInsertRow

Description

Inserts a row in the table below the row containing the insertion point. This command is available only when a table frame is active.

Syntax

TableInsertRow

See Also

TableInsertColumn

TableInsertColumn

Description

Inserts a column in the table to the right of the column containing the insertion point. This command is available only when a table frame is active.

Syntax

TableInsertColumn

See Also

TableInsertRow

TableDeleteRow

Description

Deletes the table row containing the insertion point. This command is available only when a table frame is active.

Syntax

TableDeleteRow

See Also

TableDeleteColumn

TableDeleteColumn

Description

Deletes the table column containing the insertion point. This command is available only when a table frame is active.

Syntax

TableDeleteColumn

See Also

TableDeleteRow

Tools Statements

The Tools Statements section describes the available commands on the WSWin Tools menu. The Tools Statements include

- ToolsSpellingCheck
- ToolsSpellingOptions
- ToolsThesaurus
- ToolsCustomizeMenu

ToolsSpellingCheck

Description

Starts a spelling check in the current document, using the settings specified with the ToolsSpellingOptions command. Because a spelling check is an interactive process, the spelling dialog box appears just as it does when you choose the command from the menu.

Syntax

ToolsSpellingCheck

Example

```
ToolsSpellingOptions "American English", "personal.dct", 1, 1, 0, 0, 1
ToolsSpellingCheck
```

Changes the spelling options to the American English dictionary and the personal dictionary "PERSONAL.DCT", checking all story lines from the beginning of the document, not checking words with numbers or in all uppercase letters. These options are saved in the WSW.INI file, and the spelling check is initiated.

See Also

ToolsSpellingOptions, ToolsThesaurus, ToolsGrammarCheck

ToolsSpellingOptions

Description

Changes the spelling options for the current session. The options can be saved for use in future sessions.

Syntax

ToolsSpellingOptions language$, personalDict$, checkAllStories%, fromBegin%, checkUpper%, checkNumbers%, saveSettings%

Parameters

language$ - The language dictionary to use for spelling and thesaurus. The string must match exactly the name of the dictionary is it displays in the dropdown list.

personalDict$ - The personal dictionary to store words not found in the main dictionary. You can specify a complete path. The personal dictionary is saved with the current document, and optionally in the WSW.INI file.

checkAllStories% - If TRUE, checks all stories and tables during a spelling check.

fromBegin% - If TRUE, always start the spelling check from the beginning of the document. If FALSE, start the spelling check from the insertion point. (If checkAllStories% is TRUE and fromBegin% is FALSE, then the spelling check starts from the top of the current page.)

checkUpper% - If TRUE, checks words in uppercase.

checkNumbers% - If TRUE, checks words with numbers in them.

saveSettings% - If TRUE, saves changes made in the command in the WSW.INI file.

The default for each parameter is to leave the setting unchanged.

Example

See ToolsSpellingCheck.

See Also

ToolsThesaurus

ToolsThesaurus

Description

Displays the Thesaurus dialog box for the current word. Note that the language specified in the ToolsSpellingOptions command also affects the thesaurus.

Syntax

ToolsThesaurus

Example

```
ToolsSpellingOptions "German",,,,,,
ToolsThesaurus
```

Changes the language to German so that the thesaurus will display German synonyms for the current word. (The current word must also be in German for this command to execute successfully.)

See Also

ToolsSpellingCheck, ToolsSpellingOptions, ToolsGrammarCheck

ToolsGrammarCheck

Description

Starts Correct Grammar in the current document. Because checking grammar is an interactive process, the Correct Grammar window appears just as it does when the command is chosen from the menu.

Syntax

ToolsGrammarCheck

See Also

ToolsSpellingCheck, ToolsThesaurus

ToolsBuildTOC

Description

Builds a table of contents for the current document, placing it in an external file in the WSWin Text format. You must then import the table of contents file in your document for formatting and printing.

Syntax

ToolsBuildTOC fileName$, title$, level1$, level2$, level3$, level4$, level5$, level6$, level7$, level8$, textAfter$

Parameters

fileName$ - The name of the file to use for exporting the table of contents.

title$ - The title for the table of contents.

level1$ - The name of the paragraph style to use for the first level of the table of contents.

level2$ - The name of the paragraph style to use for the second level of the table of contents.

level3$ - The name of the paragraph style to use for the third level of the table of contents.

level4$ - The name of the paragraph style to use for the fourth level of the table of contents.

level5$ - The name of the paragraph style to use for the fifth level of the table of contents.

level6$ - The name of the paragraph style to use for the sixth level of the table of contents.

level7$ - The name of the paragraph style to use for the seventh level of the table of contents.

level8$ - The name of the paragraph style to use for the eighth level of the table of contents.

textAfter$ - The text to place after each table of contents entry. The default is a tab followed by the page number.

Example

```
docName$ = GetDocName$()
docName$ = LEFT$(docName$, INSTR(docName$, ".")+1) + ".TOC"
ToolsBuildTOC docName$, "Table of Contents", "Heading 1", "Heading 2",
   "Heading 3",,,,,,
FileImport docName$, "WSWin Text",,
```

Obtains the name of the current document, changes the extension to ".TOC", builds a three-level table of contents, and imports it in the document.

See Also

ToolsBuildIndex, ToolsBuildList

ToolsBuildIndex

Description

Builds an index for the current document, placing it in an external file in the WSWin Text format. You must then import the index file in your document for formatting and printing.

Syntax

ToolsBuildIndex fileName$, title$, letterHeading%, useRange%, beforePage$, separator$, rangeText$, afterPage$

Parameters

fileName$ - The name of the file to use for exporting the index.

title$ - The title for the index.

letterHeading% - TRUE if letter headings should be used.

useRange% - TRUE if page ranges should be used.

beforePage$ - text to add before page numbers in the index. The default is a tab character.

separator$ - text to add between page numbers in the index. The default is a comma.

rangeText$ - text to add between page ranges. The default is a hyphen.

afterPage$ - text to add after the page numbers in the index. The default is an end of line character.

Example

```
docName$ = GetDocName$()
docName$ = LEFT$(docName$, INSTR(docName$, ".")+1) + ".IDX"
ToolsBuildTOC docName$, "Index", 1, 1,,,,
FileImport docName$, "WSWin Text",,
```

Obtains the name of the current document, changes the extension to ".IDX", builds the index using the defaults, and imports it in the document.

See Also

ToolsBuildTOC, ToolsBuildList

ToolsBuildList

Description

Builds an index for the current document, placing it in an external file in the WSWin Text format. You must then import the index file in your document for formatting and printing.

Syntax

ToolsBuildList fileName$, title$, listName$, textAfter$

Parameters

fileName$ - The name of the file to use for exporting the index.

title$ - The title for the index.

listName$ - The name of the list that you want to build.

textAfter$ - The text to insert after each list entry. The default is a tab character and the page number.

Example

```
docName$ = GetDocName$()
docName$ = LEFT$(docName$, INSTR(docName$, ".")+1) + ".LST"
ToolsBuildTOC docName$, "Table of Figures", "Figures",
FileImport docName$, "WSWin Text",,
```

Obtains the name of the current document, changes the extension to ".LST", builds the Figures list, and imports it in the document.

See Also

ToolsBuildTOC, ToolsBuildIndex

ToolsCustomizeMenu

Description

Adds a macro to one of the WSWin menus.

Syntax

ToolsCustomizeMenu menu$, position%, menuText$, macro$

Parameters

menu$ - The name of the menu to add the macro to. The name of the menu must match exactly the name of a WSWin menu: File, Edit, View, Insert, Style, Frame, Tools, Macro, Window, or Help.

position% - The position on the menu to change, from 1 to 10. Up to 10 items can be added to each menu. If none is specified, the macro is added to the bottom of the menu.

menuText$ - The text to display on the menu. Use an ampersand to specify the underlined character for the menu text.

macro$ - The name of the macro to execute. A path can be included with the macro name. If no path is specified, WSWin looks for the macro in the default macro directory. If no macro is specified, WSWin looks for a menu item at position% on the specified menu that matches menuText$. If it is found, then that item is deleted from the menu.

If no parameters are specified, the Customize Menus dialog box appears.

Example

```
ToolsCustomizeMenu "File",, "&Delete File", "delfile.wmc"
ToolsCustomizeMenu "Edit", 3, "F&ind Font", "findfont.wmc"
ToolsCustomizeMenu "Style", 5, "D&ouble Space",
```

Adds "Delete File" (with the D underlined) to the bottom of the File menu to run the macro DELFILE. Changes the command at the third position on the Edit menu to the specified text and macro. Deletes the "Double Space" command from the Style menu.

Information Statements

Information statements get information about the current document, perform conversions, and change the environment for WSWin. The following information statements are provided:

- BackTab
- CMY
- CountCharacters
- CountFonts
- CountFrames
- CountFrameStyles
- CountLineStyles
- CountPages
- CountStyles
- CountWords
- ExistAutoNumber
- ExistBookmark
- ExistFrameStyle
- ExistList
- ExistStyle
- From_DP$
- GetAlignment
- GetAppWindowSize
- GetCharacterFormat
- GetClipboardText
- GetCommandLine$
- GetCurrentDir$
- GetCurrentFrameStyle
- GetCurrentStyle
- GetDocName$
- GetDocWindowSize
- GetEditor
- GetFont$
- GetFrame
- GetFrameBorders
- GetFrameColor
- GetFrameColumns
- GetFrameGutterLines
- GetFrameMargins
- GetFrameOptions
- GetFrameScaling
- GetFrameSize
- GetFrameStatus
- GetFrameStyle$
- GetFrameTable
- GetFrameType
- GetLineStyle
- GetNextChar$
- GetPageNum
- GetPosition
- GetPrevChar$
- GetPrivateProfileInt
- GetPrivateProfileString
- GetProfileInt
- GetProfileString
- GetScreenSize
- GetSelection$

- GetStyle$
- GetStyleBullet
- GetStyleColor
- GetStyleDropCap
- GetStyleFont
- GetStyleNumbering
- GetStyleParagraph
- GetTextFrameType
- GetTextOffset
- GetUnits
- HSI
- Inkey$
- RGB
- SetAppWindowSize
- SetCurrentDir
- SetDocWindowSize
- SetPosition
- SoftReturn
- Return
- SplitColor
- StatusMsg
- To_DP
- Tab
- TypeText
- WritePrivateProfileString
- WriteProfileString
- WSWinCommandLine

BackTab

Description

In a table frame, a Backtab goes to the previous cell. In a graphics frame, it selects the previous object in the frame.

Syntax

BackTab

See Also

Tab

CMY

Description

Returns a 24-bit color value based on percentages of the colors cyan, magenta, and yellow.

Syntax

colorVal% = CMY (cyanval%, magval%, yellowval%)

Parameters

cyanval% - The value for cyan, from 0 to 100.

magval% - The value for magenta, from 0 to 100.

yellowval% - The value for yellow, from 0 to 100.

Returns

A 24-bit color value equivalent to the specified colors.

Example

```
backColor% = CMY(100,0,0)
borderColor% = CMY(0,100,0)
gutterColor% = CMY(0,0,100)
DefineFrameStyle "ColorFrame",
DefineFrameStyleColor backColor%, borderColor%, gutterColor%
EndDefineFrameStyle
```

Modifies the ColorFrame frame style to have a cyan background, magenta borders, and yellow gutter lines.

See Also

RGB, HSI, SplitColor

CountCharacters

Description

Counts the number of characters in the document or the current story line.

Syntax

numChars% = CountCharacters (where%)

Parameters

where% - Where to count characters: 0 = entire document, 1 = current story line, 2 = current frame, 3 = current column, 4 = current selection.

Returns

The number of characters counted.

Example

```
numChars% = CountCharacters(0)
MESSAGE "Document has " + STR$(numChars%) + " characters."
```

Counts the characters in the entire document and displays a message.

See Also

CountWords

CountFonts

Description

Counts the number of fonts currently in the system.

Syntax

numFonts% = CountFonts ( )

Returns

The number of fonts currently in the system. The value returned depends on your printer and the number of installed fonts.

Example

See GetFont$

CountFrames

Description

Counts the number of frames on the current page or in the document.

Syntax

numFrames% = CountFrames (where%, type%)

Parameters

where% - Where to count frames: 0 = entire document, 1 = current page

type% - What frames to count: 0 = all frames, 1 = text frames, 2 = graphics frames, 4 = table frames, 8 = EPS frames, 16 = OLE frames. To count multiple frame types, add the numbers together.

Returns

The number of frames counted. If where% = 0, page frames are counted as text frames. If where% = 1, the current page frame is not counted.

Example

```
numFrames% = CountFrames (1, 2+8+16)
MESSAGE "The current page has " + STR$(numFrames%) + " graphic frames."
```

Counts the number of graphic, EPS, and OLE frames on the current page and displays a message.

See Also

CountFrameStyles

CountFrameStyles

Description

Counts the number of frame styles in the current document.

Syntax

numStyles% = CountFrameStyles (type%)

Parameters

type% - What frames to count: 0 = all frames, 1 = text frames, 2 = graphics frames, 4 = table frames, 8 = EPS frames, 16 = OLE frames. To count multiple frame types, add the numbers together.

Returns

The number of frame styles of the specified type in the document.

Example

```
numStyles% = CountFrameStyle (1)
MESSAGE "The document has " + STR$(numStyles%) + " text frame styles."
```

Counts the number of text frame styles in the document and displays a message.

See Also

CountFrames, CountStyles

CountLineStyles

Description

Counts the number of line styles in the WSW.INI file.

Syntax

numLinesStyles% = CountLineStyles ( )

Returns

The number of line styles in the WSW.INI file.

Example

```
DIM lineStr$(256)
numLines% = CountLineStyles()
FOR i% = 1 to numLines%
   ' Line entries in WSW.INI start with 7
   ret% = GetPrivateProfileString("Borders", "Line"+STR$(i%+6), "",
   lineStr$(i%), "wsw.ini")
NEXT i%
```

Gets the number of line styles in the WSW.INI file and loads the values from the file into an array of strings.

See Also

GetPrivateProfileString

CountPages

Description

Counts the number of pages in the document or the current story line.

Syntax

numPages% = CountPages (where%)

Parameters

where% - Where to count pages: 0 = entire document; 1 = current story line.

Returns

The number of pages counted.

Example

```
numPages% = CountPages (0)
MESSAGE "The document has " + STR$(numPages%) + " pages."
```

Counts the number of pages in the document and displays a message.

See Also

CountFrames

CountStyles

Description

Counts the paragraph styles in the current document.

Syntax

numStyles% = CountStyles ( )

Returns

The number of paragraph styles in the document.

Example

```
numStyles% = CountStyles ()
MESSAGE "The document has " + STR$(numStyles%) + " paragraph styles."
```

Counts the paragraph styles in the document and displays a message.

See Also

CountFrameStyles

CountWords

Description

Counts the number of words in the document or story line.

Syntax

numWords% = CountWords (where%)

Parameters

where% - Where to count words: 0 = entire document, 1 = current story line, 2 = current frame, 3 = current column, 4 = current selection.

Returns

The number of words counted.

Example

```
numWords% = CountWords(4)
MESSAGE "The current selection has " + STR$(numWords%) + " words."
```

Counts the number of words in the current selection and displays a message.

See Also

CountCharacters

ExistAutoNumber

Description

Determines whether or not a specific auto number style exists in the document.

Syntax

ret% = ExistAutoNumber (autoNum$)

Parameters

autoNum$ - The name of an auto number style.

Returns

TRUE if the auto number style exists.

Example

```
IF ExistAutoNumber ("List 2") THEN
   StyleAutoNumModify "List 2",, 0, 1,,,,, ".",,
ELSE
   StyleAutoNumCreate "List 2", "List 1"
ENDIF
```

If the List 2 auto number style exists, then it is modified. Otherwise, it is created from the List 1 auto number style.

See Also

StyleAutoNumCreate, StyleAutoNumModify, StyleAutoNumDelete

ExistBookmark

Description

Determines whether or not a named bookmark exists in the document.

Syntax

ret% = ExistBookmark (bookmarkName$)

Parameters

bookmarkName$ - The name of the bookmark.

Returns

TRUE if the bookmark exists in the document.

Example

```
IF ExistBookmark ("LastEdit") THEN EditGoto 7,, "LastEdit",
```

If the bookmark LastEdit exists in the document, go to it.

See Also

InsertBookmark, InsertReference

ExistFrameStyle

Description

Determines whether or not a frame style exists in the document.

Syntax

ret% = ExistFrameStyle (styleName$)

Parameters

styleName$ - The name of a frame style.

Returns

TRUE if the frame style exists in the document.

Example

```
IF ExistFrameStyle("Text Border") THEN FrameApplyFrameStyle "Text Border"
```

See Also

ExistStyle, FrameCreateFrameStyle, FrameApplyFrameStyle, FrameDeleteFrameStyle

ExistList

Description

Determines whether a list exists in the document.

Syntax

ret% = ExistList (listName$)

Parameters

listName$ - The name of a list.

Returns

TRUE if the list exists in the document.

Example

```
IF ExistList ("Comments") THEN
   InsertListEntry "Comments", listText$, 0
ENDIF
```

If a list called Comments already exists in the document, insert another list entry in the list as specified in the variable listText$.

See Also

InsertListEntry

ExistStyle

Description

Determines whether a paragraph style exists in the document.

Syntax

ret% = ExistStyle (styleName$)

Parameters

styleName$ - The name of a paragraph style.

Returns

TRUE if the paragraph style exists in the document.

Example

```
IF NOT(ExistStyle ("Outline 6")) THEN
   StyleCreateStyle "Outline 6", "Outline 5"
ENDIF
```

If the paragraph style Outline 6 does not exist, create it from the Outline 5 paragraph style.

See Also

ExistFrameStyle, StyleCreateStyle, StyleApplyStyle, StyleDeleteStyle

From_DP$

Description

Converts a measurement, in decipoints to a string representing the measurement in a different unit.

Syntax

measurement$ = From_DP$ (decipoints%, units%)

Parameters

decipoints% - The measurement, in decipoints to convert from.

units% - The unit of measurement to convert to: 0 = inches, 1 = centimeters, 2 = points, 3 = picas.

Returns

A string representing the measurement in the specified unit of measurement.

Example

```
' Converting inches to centimeters.
inch$ = "1.00"
cm$ = From_DP$(To_DP(inch$, 0), 1)
MESSAGE inch$ + " inches = " + cm$ + " centimeters"
```

The To_DP function converts the inch string, in decipoints so that the From_DP$ function can convert the decipoints in centimeters. When the conversion is done, a message appears.

See Also

To_DP

GetAlignment

Description

Gets the alignment for the current paragraph or paragraph style.

Syntax

align% = GetAlignment (style%)

Parameters

style% - If TRUE, returns the alignment in the current paragraph style. If FALSE, returns the alignment in the current paragraph (if it has a local change). The default is FALSE.

Returns

The current alignment as requested: 0 = left, 1 = center, 2 = right, 3 = justified.

Example

```
IF GetAlignment(1) = 3 THEN
   DefineStyle ,
      DefineStyleParagraph ,,,,, 0,,,,,,,
   EndDefineStyle
ENDIF
```

If the current paragraph style is justified, modify it to make it left aligned.

See Also

DefineStyleParagraph

GetAppWindowSize

Description

Obtains the current size of the application window.

Syntax

GetAppWindowSize originX%, origintY%, width%, height%

Parameters

originX% - The horizontal position of the window.

origintY% - The vertical position of the window.

width% - The width of the window.

height% - The height of the window.

The parameters must be variables that are set to values, in pixels.

Example

```
GetAppWindowSize x%, y%, w%, h%
GetScreenSize w2%, h2%
IF (w% >= w2) AND (h% >= h2%) THEN MESSAGE "Window is full size."
```

Gets the size of the application window and the screen display to determine if the application window is maximized.

See Also

GetScreenSize, GetDocWindowSize, SetAppWindowSize, SetDocWindowSize

GetCharacterFormat

Description

Gets the current text formatting information for the selected text.

Syntax

GetCharacterFormat style%, fontName$, size%, bold%, italic%, underline%, strikeout%, strikeoutChar%

Parameters

style% - If TRUE, gets the text formatting information in the current paragraph style. If FALSE, gets the information from the current paragraph.

fontName$ - The name of the current font.

size% - The current point size, in decipoints.

bold% - The bold attribute: 0 = off, 1 = on.

italic% - The italic attribute: 0 = off, 1 = on.

underline% - The underline attribute: 0 = off, 1 = single, 2 = double.

strikeout% - The strikeout attribute: 0 = off, 1 = on.

strikeoutChar% - The ANSI code for the strikeout character.

All parameters except for style% must be variables.

Example

```
GetCharacterFormat 1, fontName$, size%, bold%, italic%, underline%,
   strikeout%, stkChar%
strikeChar$ = CHR$(stkChar%)
```

Gets the current formatting information from the paragraph style and assigns the strikeout character to a string variable for further processing.

See Also

DefineStyleFont

GetClipboardText

Description

Gets the text off the Clipboard.

Syntax

ret% = GetClipboardText (text$)

Parameters

text$ - a variable set to the text from the Clipboard

Returns

TRUE if the Clipboard contains text. The text from the Clipboard is stored in the variable passed as a parameter.

Example

```
IF GetClipboardText(text$) THEN
   EditPaste
ENDIF
```

Pastes the Clipboard contents in the document if the Clipboard contains text.

See Also

EditPaste, EditCut, EditCopy, EditPasteSpecial

GetCommandLine$

Description

Gets the command line string that was passed to WSWin. This command is most useful for the OnStart macro that performs additional processing when you start WSWin.

Syntax

commandLine$ = GetCommandLine$ ( )

Returns

The command line string that was passed to WSWin.

Example

```
' OnStart macro to load last document worked on.
cmdLine$ = GetCommandLine$()
IF cmdLine$ = "" THEN
   ret% = GetPrivateProfileString("Document History", "WorkDoc1", "",
   file$, "wsw.ini")
   FileOpen file$
ELSE
   WSWinCommandLine cmdLine$
ENDIF
```

Gets the command line. If it is empty, then open the most recently used document. Otherwise, continue processing the command line.

See Also

FileOpen, FilePrint

GetCurrentDir$

Description

Gets the current directory that was last used to open or save a file.

Syntax

dirName$ = GetCurrentDir$ ( )

Returns

The most recently accessed directory.

Example

```
dirName$ = GetCurrentDir$()
SetCurrentDir "c:\wswin\docs"
FileOpen
SetCurrentDir dirName$
```

Gets the current directory, then changes the current directory to C:\WSWIN\DOCS and displays the Open dialog box. After the dialog box is closed, restores the original directory.

See Also

SetCurrentDir

GetCurrentFrameStyle

Description

Gets the frame style for the current frame.

Syntax

GetCurrentFrameStyle styleName$, local%

Parameters

styleName$ - A variable set to the current frame style name.

local% - A variable that is set to TRUE if the current frame has local attributes.

Both parameters must be variables.

Example

```
GetCurrentFrameStyle styleName$, local%
IF local% THEN MESSAGE "Frame has local attributes."
```

Gets the current frame style information and displays a message if the local% variable is set to TRUE.

See Also

GetCurrentStyle

GetCurrentStyle

Description

Gets the current paragraph style information.

Syntax

GetCurrentStyle styleName$, local%

Parameters

styleName$ - A variable to hold the name of the current paragraph style.

local% - A variable that is set to TRUE if the current paragraph has local attributes.

Both parameters must be variables. If more than one paragraph is selected, the command returns the information about the first paragraph in the selection.

Example

```
GetCurrentStyle styleName$, local%
IF local% THEN MESSAGE "Current paragraph has local attributes."
```

Gets the style information for the current paragraph and displays a message if local attributes are present.

See Also

GetCurrentFrameStyle

GetDocName$

Description

Gets the name of the current document.

Syntax

docName$ = GetDocName$ ( )

Returns

The entire path and filename for the current document.

Example

```
' Save copy of the file to B: drive
docName$ = GetDocName$()
WHILE INSTR(docName$, "\")
   docName$ = MID$(docName$, INSTR(docName$, "\") + 1), )
WEND
FileSaveAsCopy "B:\" + docName$
```

Extracts the filename and extension for the current document and saves a copy of the file to the root directory of the drive B.

See Also

FileOpen, FileSaveAs, FileSaveAsCopy

GetDocWindowSize

Description

Obtains the current size of the document window.

Syntax

GetDocWindowSize originX%, originY%, width%, height%

Parameters

originX% - The horizontal position of the window.

originY% - The vertical position of the window.

width% - The width of the window.

height% - The height of the window.

All parameters must be variables set to values, in pixels.

Example

```
GetAppWindowSize x%, y%, w%, h%
GetDocWindowSize x2%, y2%, w2%, h2%
w2% = w% / 2
SetDocWindowSize x2%, y2%, w2%, h2%
```

Sets the document window size to half the width of the application window.

See Also

GetScreenSize, GetAppWindowSize, SetAppWindowSize, SetDocWindowSize

GetEditor

Description

Gets the currently active editor (Page or Draft) for the current document.

Syntax

editor% = GetEditor ( )

Returns

TRUE if the Page Editor is active and FALSE if the Draft Editor is active.

Example

```
IF GetEditor() THEN ViewZoomLevel 150
```

Changes the zoom level to 150% if the Page Editor is active.

See Also

ViewEditor

GetFont$

Description

Gets the name of a font from the system.

Syntax

fontName$ = Font$ (count%)

Parameters

count% - A positional parameter to determine what font to retrieve.

Returns

The name of the font at the specified position.

Example

```
DIM fonts$(256)
numFonts% = CountFonts()
FOR i% = 1 TO numFonts$
   fonts$(i%) = GetFont$(i%)
NEXT i%
```

Fills an array of strings with the names of all the fonts in the system.

Note WSWin supports a maximum of 256 font families. If you have more than 256 font families installed, you can retrieve only the first 256.

See Also

CountFonts

GetFrame

Description

Gets the number of a specific frame in the document.

Syntax

frameNum% = GetFrame (count%, where%, type%)

Parameters

count% - The position of the frame in the document. If 0, the function returns the frame number of the current frame.

where% - Where to count frames: 0 = entire document, 1 = current page.

type% - What frames to count: 0 = all frames, 1 = text frames, 2 = graphics frames, 4 = table frames, 8 = EPS frames, 16 = OLE frames. To count multiple frame types, add the numbers together.

Returns

The frame number for the specified frame.

Example

```
DIM frames%(2000)
numFrames% = CountFrames(0, 1)
FOR i% = 1 TO numFrames%
   frames%(i%) = GetFrame(i%, 0, 1)
NEXT i%
currFrame% = GetFrame(0,,)
```

Counts the number of text frames in the document and fills an array with the numbers of all the text frames. Finally, gets the frame number of the current frame.

See Also

CountFrames

GetFrameBorders

Description

Gets the border information for the current frame.

Syntax

GetFrameBorders position%, radius%, outsideSpace%, outsideThickness%, spaceBetween%, insideThickness%

Parameters

position% - The border position you want information about: 0 = none, 1 = left (inside for facing pages), 2 = right (outside for facing pages), 3 = top, 4 = bottom, 5 = all sides.

radius% - If position% = 5 (all sides), the radius of the border if rounded corners are desired, in decipoints. Radius% = 0 for no rounded corners.

outsideSpace% - If lineStyle% = 7, the space outside the lines, in decipoints.

outsideThickness% - If lineStyle% = 7, the thickness of the outer line, in decipoints.

spaceBetween% - If lineStyle% = 7, the space between the lines, in decipoints.

insideThickness% - If lineStyle% = 7, the thickness of the inner line, in decipoints.

All parameters except for position% must be variables.

Example

```
GetFrameBorders 1, radius, outSpace%, outThick%, spaceBetw%, inThick%
GetFrameBorders 2, radius, outSpace2%, outThick2%, spaceBetw2%, inThick2%
IF (outThick2% <> outThick%) OR (inThick2% <> inThick%) THEN
   MESSAGE "Lines are different."
ENDIF
```

Gets the line style information for the left and right edges of the current frame. If the lines are different, displays a message.

See Also

DefineFrameStyleBorders

GetFrameColor

Description

Gets the color information for the current frame.

Syntax

GetFrameColor backgroundColor%, borderColor%, gutterLineColor%

Parameters

backgroundColor% - A variable to hold the background color of the frame.

borderColor% - A variable to hold the border color for the frame.

gutterLineColor% - A variable to hold the gutter line color for the frame.

All parameters are 24-bit color values.

Example

```
GetFrameColor backColor%, borderColor%, gutterColor%
SplitColor backColor%, red%, green%, blue%, 0
redStr$ = "red (" + STR$(red%) + ")"
grnStr$ = "green (" + STR$(green%) + ")"
bluStr$ = "and blue (" + STR$(blue%) + ")."
MESSAGE "The background is made up of " + redStr$ + grnStr$ + bluStr$
```

Gets the colors of the frame and splits the background color in its red, green, and blue components. A message appears with information about the background color.

See Also

HSI, RGB, CMY, SplitColor

GetFrameColumns

Description

Gets the column information about the current text frame.

Syntax

GetFrameColumns numcols%, gutterWidth%, balanced%, unevenCols%, firstColWidth%

Parameters

numcols% - A variable to hold the number of columns in the frame.

gutterWidth% - A variable to hold the width of the gutter between columns.

balanced% - A variable to hold TRUE if the columns are balanced.

unevenCols% - A variable to hold TRUE if the columns are uneven widths.

firstColWidth% - If unevenCols% = TRUE, this variable will hold the width of the first column.

All parameters must be variables.

Example

```
GetFrameColumns num%, gutter%, balanced%, uneven%, first%
IF uneven% THEN
   DefineFrameStyle ,
   DefineFrameStyleColumns ,,, 0,
   EndDefineFrameStyle
ENDIF
```

Gets information about the columns in the frame. If uneven column widths is turned on, the setting is turned off.

See Also

DefineFrameStyleColumns

GetFrameGutterLines

Description

Gets the gutter line information for the current text frame.

Syntax

GetFrameGutterLines height%, inset%, leftLine%, spaceBetween%, rightLine%

Parameters

height% - A variable to hold the height of the gutter line: 0 = frame, 1 = margins, 2 = text.

inset% - A variable to hold the amount of additional space the gutter line is inset, in decipoints.

leftLine% - A variable to hold the width of the left line, in decipoints.

spaceBetween% - A variable to hold the width of the space between the lines, in decipoints.

rightLine% - A variable to hold the width of the right line, in decipoints.

Example

```
GetFrameGutterLines ht%, in%, left%, betw%, right%
IF left% <> right% THEN
   DefineFrameStyle ,
   DefineFrameStyleGutterLines ,, 7, left%,, left%
   EndDefineFrameStyle
ENDIF
```

Gets the information about the gutter lines for the current frame, and if the two lines are different, makes them both the same.

See Also

DefineFrameStyleGutterLines

GetFrameMargins

Description

Gets the information about the margins of the current frame.

Syntax

GetFrameMargins left%, right%, top%, bottom%

Parameters

left% - A variable to hold the left margin, in decipoints.

right% - A variable to hold the right margin, in decipoints.

top% - A variable to hold the top margin, in decipoints.

bottom% - A variable to hold the bottom margin, in decipoints.

Example

```
GetFrameMargins l%, r%, t%, b%
smallest% = l%
IF smallest% > r% THEN smallest% = r%
IF smallest% > t% THEN smallest% = t%
IF smallest% > b% THEN smallest% = b%
MESSAGE "The smallest margin is " + From_DP$(smallest%, 0) + "inches."
```

Determines the smallest margin for the current frame and displays a message.

See Also

DefineFrameStyleMargins, From_DP$

GetFrameOptions

Description

Gets the frame options for the current frame.

Syntax

GetFrameOptions transparent%, textWrapAround%, backgroundFill%

Parameters

transparent% - A variable to hold the transparency attribute: 0 = off, 1 = on.

textWrapAround% - A variable to hold the text wrap attribute: 0 = off, 1 = on.

backgroundFill% - A variable to hold the background fill attribute: 0 = fill to frame edge, 1 = fill to inner border, 2 = fill to outer border.

Example

```
GetFrameOptions trans%, wrap%, fill%
IF wrap% THEN MESSAGE "Text wrap is on."
```

Displays a message if the current frame has the text wrap attribute turned on.

See Also

DefineFrameStyleOptions

GetFrameScaling

Description

Gets scaling settings for the current graphic, EPS, or OLE frame.

Syntax

GetFrameScaling scaling%, horzPos%, vertPos%

Parameters

scaling% - A variable to hold the scaling attribute: 0 = crop image, 1 = scale to fit, 2 = scale to fit and maintain aspect ratio.

horzPos% - A variable to hold the horizontal position if scaling% is 2: 0 = left, 1 = center, 2 = right.

vertPos% - A variable to hold the vertical position if scaling% is 2: 0 = top, 1 = center, 2 = bottom.

Example

```
GetFrameScaling scale%, h%, v%
IF scale% = 2 THEN
   DefineFrameStyle ,
   DefineFrameStyleScaling 2, 1, 1
   EndDefineFrameStyle
ENDIF
```

Gets the scaling information for the current frame, and changes the horizontal and vertical positioning to left and top if maintain aspect ratio is turned on.

See Also

DefineFrameStyleScaling

GetFrameSize

Description

Gets the size of the current frame.

Syntax

GetFrameSize originX%, originY%, width%, height%

Parameters

originX% - A variable to hold the horizontal position of the frame, in decipoints.

originY% - A variable to hold the vertical position of the frame, in decipoints.

width% - A variable to hold the width of the frame, in decipoints.

height% - A variable to hold the height of the frame, in decipoints.

Example

```
frameNum% = GetFrame(0,, )
ParentFrame
GetFrameSize x%, y%, w%, h%
GetFrameMargins l%, r%, t%, b%
x% = x% + l%
y% = y% + t%
w% = (w% - (l% + r%)) / 2
h% = (h% - (t% + b%))
EditGoto 9, frameNum%,,
FrameSize x%, y%, w%, h%
```

Gets the number of the current frame and goes to the parent of the current frame and gets the parent's size and margins. Goes back to the child frame and makes the child frame fit inside the margins of the parent frame, filling only half the width.

See Also

FrameSize

GetFrameStatus

Description

Gets the floating status for the current frame.

Syntax

status% = GetFrameStatus ( )

Returns

TRUE if the frame is floating or FALSE if the frame is fixed.

Example

```
IF NOT(GetFrameStatus()) THEN FrameFloatFixed
```

Gets the status of the current frame, and makes the frame floating, if it isn't already.

See Also

FrameFloatFixed

GetFrameStyle$

Description

Gets the name of a frame style from the document.

Syntax

styleName$ = GetFrameStyle$ (count%, type%)

Parameters

count% - The position in the frame style list.

type% - What frames to count: 0 = all frames, 1 = text frames, 2 = graphics frames, 4 = table frames, 8 = EPS frames, 16 = OLE frames. To count multiple frame types, add the numbers together.

Returns

The name of the frame style specified by the parameters.

Example

```
DIM styles$(255)
type% = 2 + 8 + 16
numStyles% = CountFrameStyles(type%)
FOR i% = 1 TO numStyles%
   styles$(i%) = GetFrameStyle$(i%, type%)
NEXT i%
```

Counts the number of graphic, EPS, and OLE frame styles in the current document and loads their names in an array. The array can be used in a list in a dialog box.

See Also

CountFrameStyles

GetFrameTable

Description

Gets the table frame style information from the current frame.

Syntax

GetFrameTable showFirstRow%, displayNextRow%, displayNextCol%

Parameters

showFirstRow% - A variable to hold the show first row attribute: 0 = off, 1 = on.

displayNextRow% - A variable to hold the display next row attribute: 0 = off, 1 = on.

displayNextCol% - A variable to hold the display next column attribute: 0 = off, 1 = on.

Example

```
GetFrameTable first%, row%, column%
```

Loads the variables with the table frame settings for the current table frame.

See Also

DefineFrameStyleTable

GetFrameType

Description

Gets the type of the current frame (text, graphic, EPS, table, OLE).

Syntax

type% = GetFrameType ( )

Returns

The type of frame for the current frame: 0 = text, 1 = graphic, 2 = table, 3 = EPS, 4 = OLE.

Example

```
IF GetFrameType() = 0 THEN
   FrameApplyFrameStyle "Text"
ENDIF
```

Applies the Text frame style if the current frame is a text frame.

See Also

GetTextFrameType, InsertFrame

GetLineStyle

Description

Gets the specified line style from the line style list in the WSW.INI file.

Syntax

GetLineStyle count%, outsideSpace%, outsideThickness%, spaceBetween%, insideThickness%

Parameters

count% - The count in the line style list.

outsideSpace% - The space outside the lines, in decipoints.

outsideThickness% - The thickness of the outer line, in decipoints.

spaceBetween% - The space between the lines, in decipoints.

insideThickness% - The thickness of the inner line, in decipoints.

All parameters except for count% must be variables.

Example

```
DIM lines%(1000, 4)
numLines% = CountLineStyles()
FOR i% = 1 TO numLines%
   GetLineStyle i%, lines%(i%, 1), lines%(i%, 2), lines%(i%, 3), lines%(i%,
   4)
NEXT i%
```

Creates an array of line style settings and uses the GetLineStyle command to fill the items in the array.

See Also

CountLineStyles, DefineStyleLines, DefineFrameStyleBorders, TableLines

GetNextChar$

Description

Gets the character to the right of the insertion point.

Syntax

char$ = GetNextChar$ (tags%)

Parameters

tags% - If TRUE, returns the next character whether it is a tag or a regular character. If FALSE, returns only regular characters and some tags (space tags, tabs, carriage returns).

Returns

The next character or tag, depending on the value of the tags% parameter. Tags are returned as WSWin text tags. For more information, see WSWin Text.

Example

```
next$ = GetNextChar$(1)
IF LEFT$(next$, 4) = "<St " THEN
   MESSAGE "The next character is a style tag."
ENDIF
```

Gets the next character or tag and displays a message if the next character is a style tag.

See Also

GetPrevChar$, CharRight, CharLeft

GetPageNum

Description

Gets the number of the page containing the insertion point.

Syntax

pageNum% = GetPageNum (which%)

Parameters

which% - If TRUE, returns the value of the Page auto number (which can be reset by the user). If FALSE, returns the number of the physical page in the document (as displayed in the page scroll on the document window). The default is FALSE.

Returns

The page number as specified.

Example

```
page% = GetPageNum(0)
IF page% = 10 THEN
   InsertAutoNumber "Page",, 1, 1, 0
ENDIF
```

Gets the page number of the current page, and if the page number is 10, inserts a page number reset tag to start the page numbering with 1.

See Also

InsertAutoNumber

GetPosition

Description

Gets the current position of the insertion point on the screen.

Syntax

GetPosition x%, y%

Parameters

x% - A variable to hold the horizontal position of the insertion point.

y% - A variable to hold the vertical position of the insertion point.

Both values are screen coordinates.

Example

```
GetPosition x%, y%
SetPosition x%+10, y%+50
Moves the insertion point to the screen location ten pixels to the right
   and 50 pixels down from its current location.
```

See Also

SetPosition

GetPrevChar$

Description

Gets the character to the left of the insertion point.

Syntax

char$ = GetPrevChar$ (tags%)

Parameters

tags% - If TRUE, returns the next character whether it is a tag or a regular character. If FALSE, returns only regular characters and some tags (space tags, tabs, carriage returns).

Returns

The previous character or tag, depending on the value of the tags% parameter. Tags are returned as WSWin text tags. For more information, see WSWin Text.

Example

```
prev$ = GetPrevChar$(0)
IF LEFT$(prev$, 4) = "<Sp " OR prev$ = " " OR prev$ = <Tb> THEN
   MESSAGE "The previous character is a space or a tab."
ENDIF
```

Gets the previous character and displays a message if the character is a space or a tab (including the special space characters).

See Also

GetNextChar$, CharRight, CharLeft

GetPrivateProfileInt

Description

Gets an integer value from an entry in an INI file.

Syntax

ret% = GetPrivateProfileInt (appName$, key$, default%, fileName$)

Parameters

appName$ - The name of the section of the INI file. Sections in the INI file are denoted with square brackets, but you don't need to specify them in the string.

key$ - The key for the entry (the string to the left of the equal sign). The equal sign is not necessary.

default% - The default value if the entry isn't found.

fileName$ - The name of the INI file to search.

Returns

The integer value found for the key, or the default value if not found.

Example

```
ruler% = GetPrivateProfileInt("Preferences", "Ruler", 1, "wsw.ini")
```

Gets the value of the Ruler setting in the WSW.INI file. If it is not found, the default value is 1.

See Also

GetPrivateProfileString, GetProfileInt

GetPrivateProfileString

Description

Gets a string value from an entry in an INI file.

Syntax

ret% = GetPrivateProfileString (appName$, key$, default$, returnString$, fileName$)

Parameters

appName$ - The name of the section of the INI file. Sections in the INI file are denoted with square brackets, but you don't need to specify them in the string.

key$ - The key for the entry (the string to the left of the equal sign). The equal sign is not necessary.

default$ - The default string if the entry isn't found.

returnString$ - A variable to hold the string, or the default value if the entry isn't found.

fileName$ - The name of the INI file to search.

Returns

The number of characters in returnString$.

Example

```
ret% = GetPrivateProfileString("Preferences", "Template", "", template$,
   "wsw.ini")
```

Sets the variable template$ to the name of the default template.

See Also

GetPrivateProfileInt, GetProfileString, WritePrivateProfileString

GetProfileInt

Description

Gets an integer value from an entry in the WIN.INI file.

Syntax

GetProfileInt (appName$, key$, default%)

Parameters

appName$ - The name of the section of the INI file. Sections in the INI file are denoted with square brackets, but you don't need to specify them in the string.

key$ - The key for the entry (the string to the left of the equal sign). The equal sign is not necessary.

default% - The default value if the entry isn't found.

Returns

The integer value found for the key, or the default value if it is not found.

Example

```
speed% = GetProfileInt("windows", "DoubleClickSpeed", 400)
```

Gets the value of the double-click speed in the WIN.INI file.

See Also

GetPrivateProfileInt, GetProfileString

GetProfileString

Description

Gets a string value from an entry in the WIN.INI file.

Syntax

ret% = GetProfileString (appName$, key$, default$, returnString$)

Parameters

appName$ - The name of the section of the INI file. Sections in an INI file are denoted with square brackets, but you don't need to specify them in the string.

key$ - The key for the entry (the string to the left of the equal sign). The equal sign is not necessary.

default$ - The default string if the entry isn't found.

returnString$ - A variable to hold the string, or the default value if the entry isn't found.

Returns

The number of characters in returnString$.

Example

```
ret% = GetProfileString("windows", "Programs", "exe com pif bat",
   progList$)
```

Sets the variable progList$ to the programs setting in the WIN.INI file.

See Also

GetPrivateProfileString, GetProfileInt, WriteProfileString

GetScreenSize

Description

Gets the current display dimensions.

Syntax

GetScreenSize width%, height%

Parameters

width% - The width of the screen, in pixels.

height% - The height of the screen, in pixels.

Both parameters must be variables.

Example

```
GetScreenSize width%, height%
IF height% > 480 THEN
   MESSAGE "This is a high-resolution display."
ENDIF
```

Displays a message if the screen height is larger than 480 pixels.

See Also

GetAppWindowSize, GetDocWindowSize

GetSelection$

Description

Gets the current text selection.

Syntax

sel$ = GetSelection$ (ascii%)

Parameters

ascii% - If TRUE, gets only the regular characters in the selection. If FALSE, gets the selection in WSWin text format. The default is FALSE.

Returns

The selected text in the specified format. If no text is selected, returns an empty string.

Example

```
sel$ = GetSelection$()
IF sel$ <> "" THEN
   EditDelete
   TypeText sel$
ENDIF
```

Gets the current selection, deletes it and inserts it again.

See Also

GetTextOffset, TypeText

GetStyle$

Description

Gets the name of a paragraph style.

Syntax

styleName$ = GetStyle$ (count%)

Parameters

count% - The position of the character style in the document's style list.

Example

```
DIM styles$(255)
numStyles% = CountStyles()
FOR i% = 1 TO numStyles%
   styles$(i%) = GetStyle$(i%)
NEXT i%
```

Gets the names of all the styles in the document and stores them in an array. The array can then be used in a list in a dialog box.

See Also

GetFrameStyle$, GetCurrentStyle

GetStyleBullet

Description

Gets the bullet characteristics of the current paragraph.

Syntax

GetStyleBullet bulletType%, filled%, position%, on%

Parameters

bulletType% - A variable to hold the bullet type: 0 = none, 1 = round, 2 = square, 3 = hyphen.

filled% - A variable to hold whether the bullet is filled or not.

position% - A variable to hold the bullet position, in decipoints.

on% - A variable that is set to TRUE if the bullets are on in the paragraph.

Example

```
GetStyleBullet type%, filled%, position%, on%
IF NOT(on%) THEN
   DefineStyle ,
   DefineStyleBullet 1, filled%, position%
   EndDefineStyle
ENDIF
Gets the current bullet style, and if it isn't on, turns it on.
```

See Also

DefineStyleBullet

GetStyleColor

Description

Gets the color information for the current paragraph.

Syntax

GetStyleColor textColor%, lineColor%, effectColor%

Parameters

textColor% - A variable to hold the current text color.

lineColor% - A variable to hold the current line color.

effectColor% - A variable to hold the current effect color (for bullets or drop cap).

All variables will receive 24-bit color values.

Example

```
GetStyleColor textColor%, lineColor%, effectColor%
SplitColor textColor%, hue%, satur%, intens%, 2
```

Gets the text color for the current paragraph and splits it in hue, saturation, and intensity values.

See Also

DefineStyleColor, HSI, CMY, RGB, SplitColor

GetStyleDropCap

Description

Gets the drop cap settings for the current paragraph.

Syntax

GetStyleDropCap typeFace$, size%, lineHeight%, bold%, italic%, alignBaseline%, numCharacters%, adjustBaseline%, on%

Parameters

typeFace$ - A variable to hold the font for the drop cap.

size% - A variable to hold the point size for the drop cap, in decipoints.

lineHeight% - A variable to hold the line height for the drop cap, in decipoints.

bold% - A variable to hold the bold attribute for the drop cap: 0 = off, 1 = on.

italic% - A variable to hold the italic attribute for the drop cap: 0 = off, 1 = on.

alignBaseline% - A variable to hold the alignment setting for the drop cap: 0 = align with top of characters, 1 = align with nearest baseline.

numCharacters% - A variable to hold the number of characters for the drop cap: from 1 to 9.

adjustBaseline% - A variable to hold the additional baseline adjust setting, in decipoints.

on% - A variable that is set to TRUE if the drop cap setting is on in the paragraph.

Example

```
GetStyleDropCap font$, size%, ht%, b%, i%, align%, num%, adjust%, on%
```

Gets the drop cap information for the current paragraph and stores it in the variables.

See Also

DefineStyleDropCap

GetStyleFont

Description

Gets the font information for the current paragraph style. This command is similar to GetCharacterFormat, except that it gets the information for the style (local or global) of the current paragraph, rather than the text in the paragraph (which may be affected by tags in the text).

Syntax

GetStyleFont typeFace$, size%, lineHeight%, bold%, italic%, underline%, strikeout%, hidden%, placeUnderline%, strikeoutChar$

Parameters

typeFace$ - A variable to hold the font for the paragraph style.

size% - A variable to hold the point size for the paragraph style.

lineHeight% - A variable to hold the line height for the paragraph style.

bold% - A variable to hold the bold attribute for the paragraph style.

italic% - A variable to hold the italic attribute for the paragraph style.

underline% - A variable to hold the underline for the paragraph style: 0 = none, 1 = single, 2 = double.

strikeout% - A variable to hold the strikeout attribute for the paragraph style.

hidden% - A variable to hold the hidden for the paragraph style.

placeUnderline% - A variable to hold the underline position for the paragraph style: 0 = at baseline, 1 = below descenders.

strikeoutChar$ - A variable to hold the strikeout character for the paragraph style.

Example

```
GetStyleFont font$, size%, ht%, bd%, it%, ul%, so%, hd%, place%, char$
```

Gets the font information in the current paragraph style and places it in the variables.

See Also

DefineStyleFont

GetStyleNumbering

Description

Gets the numbering information for the current paragraph style.

Syntax

GetStyleNumbering numberStyle$, level%, position%, alignment%, on%

Parameters

numberStyle$ - A variable to hold the auto number style for the paragraph style.

level% - A variable to hold the level for the auto number, if the auto number is Outline.

position% - A variable to hold the position for the auto number, in decipoints.

alignment% - A variable to hold the line height for the paragraph style.

on% - A variable set to on% if numbering is on in the paragraph style.

Example

```
GetStyleNumbering numStyle$, level%, pos%, align%, on%
```

Gets the numbering for the current paragraph style.

See Also

DefineStyleNumbering

GetStyleParagraph

Description

Gets the paragraph formatting for the current paragraph style.

Syntax

GetStyleParagraph firstIndent%, leftIndent%, rightIndent%, spaceAbove%, spaceBelow%, alignment%, location%, spanColumns%, allowHyphenBreak%, numHyphen%, keepWithNext%, allowBreakWithin%, widowControl%

Parameters

firstIndent% - A variable to hold the first indent setting, in decipoints.

leftIndent% - A variable to hold the left indent setting, in decipoints.

rightIndent% - A variable to hold the right indent setting, in decipoints.

spaceAbove% - A variable to hold the space above setting, in decipoints.

spaceBelow% - A variable to hold the space below setting, in decipoints.

alignment% - A variable to hold the alignment: 0 = left, 1 = center, 2 = right, 3 = justified.

location% - A variable to hold the paragraph location: 0 = normal, 1 = top of frame, 2 = top of column.

spanColumns% - A variable set to TRUE if span columns is turned on.

allowHyphenBreak% - A variable set to TRUE if auto hyphenation is turned on.

numHyphen% - A variable to hold the number of consecutive hyphens allowed.

keepWithNext% - A variable set to TRUE if Keep with Next setting is on.

allowBreakWithin% - A variable set to TRUE if Allow Break Within setting is on.

widowControl% - A variable set to TRUE if widow control is on.

Example

```
GetStyleParagraph first%, left%, right%, above%, below%, align%, loc%,
   span%, hyphen%, numHyph%, keep%, break%, widow%
IF first% = left% THEN
   DefineStyle ,
   DefineStyleParagraph first% + 360,,,,,,,,,,,
   EndDefineStyle
ENDIF
```

Gets the paragraph settings for the current paragraph. If the first indent is the same as the left indent, indent the first indent one-half inch.

See Also

DefineStyleParagraph

GetTextFrameType

Description

Gets the type of the current text frame.

Syntax

frameType% = GetTextFrameType ( )

Returns

The frame type for the current frame: 0 = not a text frame, 1 = portrait page, 2 = landscape page, 3 = dependent frame, 4 = bottom caption, 5 = top caption, 6 = left caption, 7 = right caption, 8 = header frame, 9 = footer frame

Example

```
type% = GetFrameType()
IF type% = 0 THEN    ' it's a text frame
   type% = GetTextFrameType()
   IF type% = 1 THEN ' it's a portrait page
      FramePageOrientation 1     ' make it landscape
   ENDIF
ENDIF
Finds out if the current page is a portrait page, and if so, changes it to
   landscape.
```

See Also

GetFrameType

GetTextOffset

Description

Gets the current position, relative to the beginning of the story line, of the insertion point or selected text.

Syntax

sel% = GetTextOffset (start%, end%)

Parameters

start% - A variable to hold the starting position of the text selection.

end% - A variable to hold the ending poisition of the text selection.

If no text is selected, both start% and end% have the same value.

Returns

TRUE if text is selected.

Example

```
sel% = GetTextOffset (start%, end%)
EndOfStory
EditGotoOffset start,
IF sel% THEN EditGotoOffset end, 1
```

Gets the current selection positions and goes to the end of the story line. Goes back to the start of the selection and selects to the end.

See Also

EditGotoOffset

GetUnits

Description

Gets the current unit of measurement for the document.

Syntax

units% = GetUnits ( )

Returns

The current unit of measurement: 0 = inches, 1 = centimeters, 2 = points, 3 = picas.

Example

```
measurement$ = "1.00"
units% = GetUnits()
decipoints% = To_DP(measurement$, units%)
```

See Also

From_DP$, To_DP

HSI

Description

Converts values for hue, saturation, and intensity to a 24-bit color value.

Syntax

colorVal% = HSI (hueval%, satval%, intval%)

Parameters

hueval% - The hue value, from 0 to 359. 0 = red, 60 = yellow, 120 = green, 180 = cyan, 240 = blue, 300 = magenta.

satval% - The saturation of color, from 0 to 100.

intval% - The intensity of color, from 0 to 100.

Returns

The 24-bit color value for the specified values.

Example

```
color% = HSI(0, 100, 75)
color2% = HSI(180, 100, 100)
```

Sets the variable color% to 75% red and color2 to 100% cyan.

See Also

RGB, CMY, SplitColor.

Inkey$

Description

Gets the key pressed by the user.

Syntax

char$ = Inkey$ ( )

Returns

The character corresponding to the key pressed by the user. If no key was pressed, the function returns an empty string.

Example

```
WHILE char$ <> ""
   char$ = Inkey$()
WEND
MESSAGE "You pressed:  " + char$
```

Inkey$ works by looking for a keystroke in the input buffer. If no keystroke is found, the function returns an empty string. To make sure that a key is pressed, use a loop to wait for the keystroke, as shown in the example.

See Also

INPUTBOX

Return

Description

Inserts a carriage return followed by a style tag.

Syntax

Return

Example

```
TypeText "This is some text."
Return
TypeText "This is a new paragraph."
```

Inserts some text, then inserts a carriage return before adding more text. This is often used to continue a bulleted or numbered paragraph without adding another bullet or number.

To insert a carriage return without a style tag, use SoftReturn.

See Also

TypeText

RGB

Description

Converts values for red, green, and blue to a 24-bit color value.

Syntax

colorVal% = RGB (redval%, greenval%, blueval%)

Parameters

redval% - The red value, from 0 to 255.

greenval% - The green value, from 0 to 255.

blueval% - The blue value, from 0 to 255.

Returns

The 24-bit color value based on the specified values.

Example

```
StyleTextColor RGB(0, 128, 0)
```

Changes the text color to dark green.

See Also

CMY, HSI, SplitColor

SetAppWindowSize

Description

Sets the size of the application window.

Syntax

SetAppWindowSize sizeType%, originX%, originY%, width%, height%

Parameters

sizeType% - The window type: 0 = minimized, 1 = maximized, 2 = restored, 3 = specific size.

originX% - If sizeType% = 3, the horizontal position of the window, in pixels.

originY% - If sizeType% = 3, the vertical position of the window, in pixels.

width% - If sizeType% = 3, the width of the window, in pixels.

height% - If sizeType% = 3, the height of the window, in pixels.

Example

```
GetScreenSize w%, h%
SetAppWindowSize 3, 0, 0, w%/2, h%/2
```

Position the application window in the upper-left quarter of the screen.

See Also

SetDocWindowSize, GetAppWindowSize, GetScreenSize

SetCurrentDir

Description

Sets the current directory for the next WSWin file operation.

Syntax

ret% = SetCurrentDir (currentDir$)

Parameters

currentDir$ - The name of the directory to become current.

Returns

TRUE if the command was successful.

Example

See GetCurrentDir.

See Also

FileOpen, FileNew, FileSaveAs

SetDocWindowSize

Description

Sets the position and size of the current document window.

Syntax

SetDocWindowSize sizeType%, originX%, originY%, width%, height%

Parameters

sizeType% - The window type: 0 = minimized, 1 = maximized, 2 = restored, 3 = specific size.

originX% - If sizeType% = 3, the horizontal position of the window, in pixels.

originY% - If sizeType% = 3, the vertical position of the window, in pixels.

width% - If sizeType% = 3, the width of the window, in pixels.

height% - If sizeType% = 3, the height of the window, in pixels.

Example

```
GetAppWindowSize x%, y%, h%, w%
GetDocWindowSize x2%, y2%, h2%, w2%
SetDocWindowSize x2%, y2%, h% - 80, w% - 80
Changes the document window to be 80 pixels narrower and shorter than the
   application window.
```

See Also

SetAppWindowSize, GetDocWindowSize

SetPosition

Description

Sets the position of the insertion point in device units. This command simulates a mouse click the screen.

Syntax

SetPosition x%, y%, modifier%

Parameters

x% - The horizontal position for the insertion point, in device units.

y% - The vertical position for the insertion point, in device units.

modifier% - The modifier key, if any, that modifies the action: 0 = no modifier, 4 = Shift key, 8 = Ctrl key. The default is 0.

Example

```
SetPosition 100, 100, 0
SetPosition 200, 200, 4
```

Simulates clicking the mouse at the coordinates 100, 100 and clicking again at 200, 200 with the Shift key held down.

See Also

GetPosition

SoftReturn

Description

Insert a carriage return without a style tag.

Syntax

SoftReturn

Example

```
TypeText "This is some text."
SoftReturn
TypeText "This is a new paragraph without a style tag."
```

Inserts some text followed by a carriage return without the style tag before adding more text. This is often used to continue a bulleted or numbered paragraph without adding another bullet or number.

See Also

TypeText

SplitColor

Description

Splits a 24-bit color value in its components.

Syntax

SplitColor colorval%, color1%, color2%, color3%, model%

Parameters

colorval% - The 24-bit color value to convert.

color1% - A variable to hold the first color component value (hue, red, or cyan).

color2% - A variable to hold the second color component value (saturation, green, or magenta).

color3% - A variable to hold the third color component value (intensity, green, or yellow).

model% - The color model to use: 0 = RGB, 1 = CMY, 2 = HSI.

Example

```
GetStyleColor textColor%, lineColor%, effectColor%
SplitColor textColor%, red%, green%, blue%, 0
SplitColor textColor%, cyan%, magenta%, yellow%, 1
SplitColor textColor%, hue%, sat%, int%, 2
```

Gets the text color for the current paragraph style and converts it in its component colors for all three color models.

See Also

CMY, HSI, RGB

StatusMsg

Description

Display a message on the Status Bar. The message remains until another message replaces it.

Syntax

StatusMsg msg$

Parameters

msg$ - The message to display in the Status Bar.

Example

```
msg$ = "Please type a character: "
StatusMsg msg$
WHILE char$ <> ""
   char$ = Inkey$()
WEND
msg$ = msg$ + char$
StatusMsg msg$
```

Displays a message on the Status Bar and waits for a keystroke. When the keystroke is received, adds it to the Status Bar message.

See Also

MESSAGE, MESSAGEBOX

Tab

Description

Inserts a tab character in the text. In a table frame, goes to the next cell. In a graphics frame, selects the next object in the frame.

Syntax

Tab

Example

```
Tab
TypeText "<Tb>"
```

These two lines produce the same result in a text frame.

See Also

TypeText

To_DP

Description

Converts a string representing a decimal measurement value, in decipoints.

Syntax

decipoints% = To_DP (measurement$, units%)

Parameters

measurement$ - A string representing a measurement.

units% - The unit of measurement for the string: 0 = inches, 1 = centimeters, 2 = points, 3 = picas.

Returns

The value, in decipoints of the specified measurement.

Example

```
decipoints% = To_DP("2.5", 1)
inches$ = From_DP$(decipoints%, 0)
```

Converts 2.5 centimeters, in decipoints and converts the result in inches.

See Also

From_DP$

TypeText

Description

Inserts text in the document at the insertion point.

Syntax

TypeText string$

Parameters

string$ - A string of text to insert. You can use WSWin text conventions.

Example

```
TypeText "This is a <Bd 1>test<Bd 0>.<Cr>"
```

Inserts the string "This is a **test**." followed by a carriage return. Note the use of the bold tags in the string.

See Also

GetSelection$

WritePrivateProfileString

Description

Writes a string in an INI file.

Syntax

ret% = WritePrivateProfileString (appName$, key$, default$, fileName$)

Parameters

appName$ - The name of the section of the INI file. Sections in the INI file are denoted with square brackets, but you don't need to specify them in the string.

key$ - The key for the entry (the string to the left of the equal sign). The equal sign is not necessary.

default$ - The string to write to the INI file after the equal sign. The equal sign is not necessary.

fileName$ - The name of the INI file to write to.

Returns

TRUE if the function was successful.

Example

```
ret% = WritePrivateProfileString ("Preferences", "Template", "def.wst",
   "wsw.ini")
```

Changes the default template in the INI file to DEF.WST.

See Also

GetPrivateProfileString, GetPrivateProfileInt, WriteProfileString

WriteProfileString

Description

Writes a string in the WIN.INI file.

Syntax

ret% = WriteProfileString (appName$, key$, default$)

Parameters

appName$ - The name of the section of the INI file. Sections in an INI file are denoted with square brackets, but you don't need to specify them in the string.

key$ - The key for the entry (the string to the left of the equal sign). The equal sign is not necessary.

default$ - The string to write to the INI file after the equal sign. The equal sign is not necessary.

Returns

TRUE if the function was successful.

Example

```
ret% = WriteProfileString ("windows", "DoubleClickSpeed", 500)
```

Changes the value of the double-click speed in the WIN.INI file.

See Also

GetProfileString, GetProfileInt, WritePrivateProfileString

WSWinCommandLine

Description

Sends a command line string to WSWin for further processing.

Syntax

WSWinCommandLine commandLine$

Parameters

commandLine$ - The command line string for WSWin to handle.

Example

See GetCommandLine$

Navigation Statements

Use navigation statements to move the insertion point to different locations in a document. When you move the insertion point by characters, words, sentences, and paragraphs, the movement is cumulative. For example, you can use two WordLeft commands to move the insertion point to the beginning of the current word and then to the beginning of the previous word. Keep in mind when you write a macro that the insertion point is not always at the beginning of a word or sentence.

The following navigation statements are available in the WSWin macro language:

- CharLeft
- CharRight
- ChildFrame
- DeleteChar
- DeleteLine
- DeleteLineLeft
- DeleteLineRight
- DeleteWordLeft
- DeleteWordRight
- EndOfCol
- EndOfDoc
- EndOfFrame
- EndOfLine
- EndOfPara
- EndOfStory
- LineDown
- LineUp
- NextDocWindow
- NextFrame
- NextLinkedFrame
- PageDown
- PageUp
- ParaDown
- ParaUp
- ParentFrame
- PrevDocWindow
- PrevFrame
- PrevLinkedFrame
- SentenceDown
- SentenceUp
- StartOfCol
- StartOfDoc
- StartOfFrame
- StartOfLine
- StartOfPara
- StartOfStory
- WordLeft
- WordRight

CharLeft

Description

Moves the insertion point a number of character positions to the left.

Syntax

CharLeft count%, select%, tags%

Parameters

count% - The number of character positions to move. The default is 1.

select% - If TRUE, the selection will be extended. The default is FALSE.

tags% - If TRUE, tags will be treated as individual characters, allowing greater precision. The default is FALSE.

Example

```
CharLeft 1,,
CharLeft 1,, 1
CharLeft 3, 1,
```

Moves one character to the left, and then moves one character or tag position to the left. Finally, moves three characters to the left while extending the insertion point.

Note Like the arrow keys, CharLeft extends the selection until you use a character movement command that does not extend the selection.

See Also

CharRight, WordLeft

CharRight

Description

Moves the insertion point a number of character positions to the right.

Syntax

CharRight count%, select%, tags%

Parameters

count% - The number of character positions to move. The default is 1.

select% - If TRUE, the selection will be extended. The default is FALSE.

tags% - If TRUE, tags will be treated as individual characters, allowing greater precision. The default is FALSE.

Example

```
CharRight 1,,
CharRight 1,, 1
CharRight 3, 1,
```

Moves one character to the right, and then moves one character or tag position to the right. Finally, moves three characters to the right while extending the insertion point.

Note Like the arrow keys, CharLeft extends the selection until you use a character movement command that does not extend the selection.

See Also

CharLeft, WordRight

ChildFrame

Description

Goes to the first child frame owned by the current frame.

Syntax

ChildFrame

Example

See ParentFrame

See Also

NextFrame, PrevFrame

DeleteChar

Description

Deletes the character to the right of the insertion point.

Syntax

DeleteChar

Example

```
DeleteChar
EditDelete 0,
```

You can use either of the above commands to perform the same function.

See Also

EditDelete

DeleteLine

Description

Deletes the current line of text.

Syntax

DeleteLine

See Also

EditDelete, DeleteLineLeft, DeleteLineRight

DeleteLineLeft

Description

Deletes the portion of the current line to the left of the insertion point.

Syntax

DeleteLineLeft

Example

```
DeleteLineLeft
EditDelete 5,
```

You can use either of the above commands to perform the same function.

See Also

EditDelete, DeleteLine, DeleteLineRight

DeleteLineRight

Description

Deletes the portion of the current line to the right of the insertion point.

Syntax

DeleteLineRight

Example

```
DeleteLineRight
EditDelete 2,
```

You can use either of the above commands to perform the same function.

See Also

EditDelete, DeleteLine, DeleteLineLeft

DeleteWordLeft

Description

Deletes the word to the left of the insertion point.

Syntax

DeleteWordLeft

Example

```
DeleteWordLeft
EditDelete 4,
```

You can use either of the above commands to perform the same function.

See Also

EditDelete, DeleteWordRight

DeleteWordRight

Description

Deletes the word to the right of the insertion point.

Syntax

DeleteWordRight

Example

```
DeleteWordRight
EditDelete 1,
```

You can use either of the above commands to perform the same function.

See Also

EditDelete, DeleteWordLeft

EndOfCol

Description

Moves the insertion point to the end of the current column.

Syntax

EndOfCol select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
StartOfCol
EndOfCol 1
```

These two commands select all the text in the current column. Use this command for text frames.

See Also

StartOfCol

EndOfDoc

Description

Moves the insertion point to the end of the document.

Syntax

EndOfDoc

Because the document may contain multiple story lines, you cannot select text with this command. To select the text in a story line, use the EndOfStory command.

See Also

StartOfDoc, EndOfStory

EndOfFrame

Description

Moves the insertion point to the end of the current text frame.

Syntax

EndOfFrame select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
StartOfFrame
EndOfFrame 1
```

Selects all the text in the current frame.

See Also

StartOfFrame

EndOfLine

Description

Moves the insertion point to the end of the current line of text.

Syntax

EndOfLine select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
StartOfLine
EndOfLine 1
```

Selects the text on the current line.

See Also

StartOfLine

EndOfPara

Description

Moves the insertion point to the end of the current paragraph (after the carriage return).

Syntax

EndOfPara select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
StartOfPara
EndOfPara 1
```

Selects the text in the current paragraph.

See Also

StartOfPara, ParaUp, ParaDown

EndOfStory

Description

Moves the insertion point to the end of the current story line.

Syntax

EndOfStory select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
StartOfStory
EndOfStory 1
```

Selects the text in the current story line.

See Also

StartOfStory

LineDown

Description

Moves the insertion point a specified number of lines down.

Syntax

LineDown count%, select%

Parameters

count% - The number of lines to move down. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
LineDown 3, 1
```

Extends the selection down three lines.

See Also

LineUp

LineUp

Description

Moves the insertion point a specified number of lines up.

Syntax

LineUp count%, select%

Parameters

count% - The number of lines to move up. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
LineUp 3, 1
```

Extends the selection up three lines.

See Also

LineDown

NextDocWindow

Description

Activates the next document window.

Syntax

NextDocWindow

See Also

PrevDocWindow

NextFrame

Description

Goes to the next frame that has the same parent.

Syntax

NextFrame

See Also

PrevFrame, ChildFrame, ParentFrame

NextLinkedFrame

Description

Goes to the next frame in the same linked chain as the current frame.

Syntax

NextLinkedFrame

See Also

PrevLinkedFrame

PageDown

Description

Goes to the next page (or pages) in the document.

Syntax

PageDown count%, select%

Parameters

count% - The number of pages to move ahead. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
PageDown 1,
```

Goes to the next page in the document.

See Also

PageUp

PageUp

Description

Goes to the previous page (or pages) in the document.

Syntax

PageUp count%, select%

Parameters

count% - The number of pages to move ahead. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
PageUp 2,
```

Goes back two pages in the document.

See Also

PageDown

ParaDown

Description

Moves the insertion point down a number of paragraphs.

Syntax

ParaDown count%, select%

Parameters

count% - The number of paragraphs to move down. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
ParaDown 3, 1
```

Extends the selection down three paragraphs.

See Also

ParaUp, EndOfPara

ParaUp

Description

Moves the insertion point up a number of paragraphs.

Syntax

ParaUp count%, select%

Parameters

count% - The number of paragraphs to move up. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
CharRight 1,,
ParaUp 1,
ParaDown 1, 1
```

Selects the entire paragraph. The CharRight command moves one character in the paragraph to make sure that the insertion point is inside the paragraph. Otherwise, ParaUp could move the insertion point to the beginning of the previous paragraph.

See Also

ParaDown, StartOfPara

ParentFrame

Description

Goes to the parent of the current frame. If the insertion point is on the page frame, this command does nothing.

Syntax

ParentFrame

Example

```
type% = GetTextFrameType()
IF type% = 0 OR type% = 3 THEN
   ParentFrame
ELSE
   ChildFrame
ENDIF
```

Goes to the parent frame if the current frame is not a text frame or is a dependent text frame. Otherwise, goes to the first child frame of the current frame.

See Also

ChildFrame, NextFrame, PrevFrame

PrevDocWindow

Description

Activates the previous document window.

Syntax

PrevDocWindow

See Also

NextDocWindow

PrevFrame

Description

Goes to the previous frame that has the same parent as the current frame.

Syntax

PrevFrame

See Also

NextFrame, ChildFrame, ParentFrame

PrevLinkedFrame

Description

Goes to the previous frame in the same linked chain as the current frame.

Syntax

PrevLinkedFrame

See Also

NextLinkedFrame

SentenceDown

Description

Moves the insertion point down a number of sentences.

Syntax

SentenceDown count%, select%

Parameters

count% - The number of sentences to move down. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
SentenceUp 1,
Sentence Down 1, 1
```

Selects the current sentence.

See Also

SentenceUp

SentenceUp

Description

Moves the insertion point up a number of sentences.

Syntax

SentenceUp count%, select%

Parameters

count% - The number of sentences to move up. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

See SentenceDown

StartOfCol

Description

Moves the insertion point to the beginning of the current column in the text frame.

Syntax

StartOfCol select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

See EndOfCol

StartOfDoc

Description

Moves the insertion point to the beginning of the document. Because the document may have multiple story lines, you cannot select text with this command. Use StartOfStory if you want to select text to the beginning of the story.

Syntax

StartOfDoc

See Also

EndOfDoc

StartOfFrame

Description

Moves the insertion point to the beginning of the current frame.

Syntax

StartOfFrame select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

See EndOfFrame

StartOfLine

Description

Moves the insertion point to the beginning of the line.

Syntax

StartOfLine select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

See EndOfLine

StartOfPara

Description

Moves the insertion point to the beginning of the paragraph.

Syntax

StartOfPara select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

See EndOfPara

See Also

ParaUp

StartOfStory

Description

Moves the insertion point to the beginning of the current story line.

Syntax

StartOfStory select%

Parameters

select% - If TRUE, extends the selection. The default is FALSE.

Example

See EndOfStory

See Also

StartOfDoc

WordLeft

Description

Moves the insertion point to the left one or more words.

Syntax

WordLeft count%, select%

Parameters

count% - The number of words to move over. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

```
WordLeft 1,
WordRight 1, 1
```

Selects the current word.

See Also

WordRight, CharLeft

WordRight

Description

Moves the insertion point to the right one or more words.

Syntax

WordRight count%, select%

Parameters

count% - The number of words to move over. The default is 1.

select% - If TRUE, extends the selection. The default is FALSE.

Example

See WordLeft

See Also

CharRight