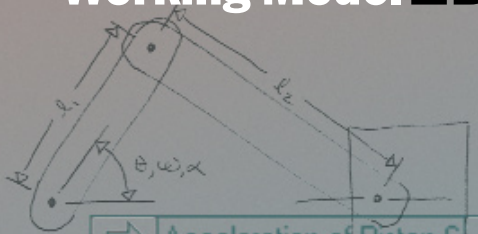
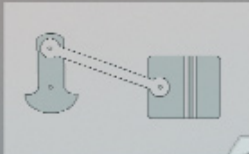
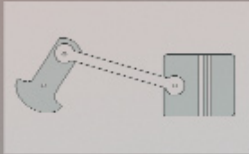
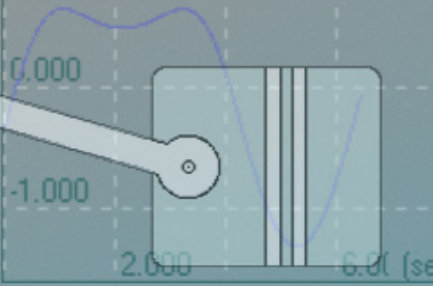


Working Model[®] 2D



Acceleration of Piston 6

(in/sec²)



X-Position of Piston 6

(in)

4.000

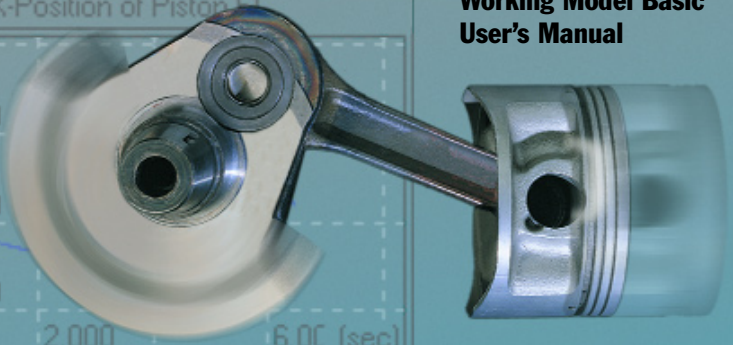
2.000

0.000

2.000

6.00 (sec)

Working Model Basic[™]
User's Manual

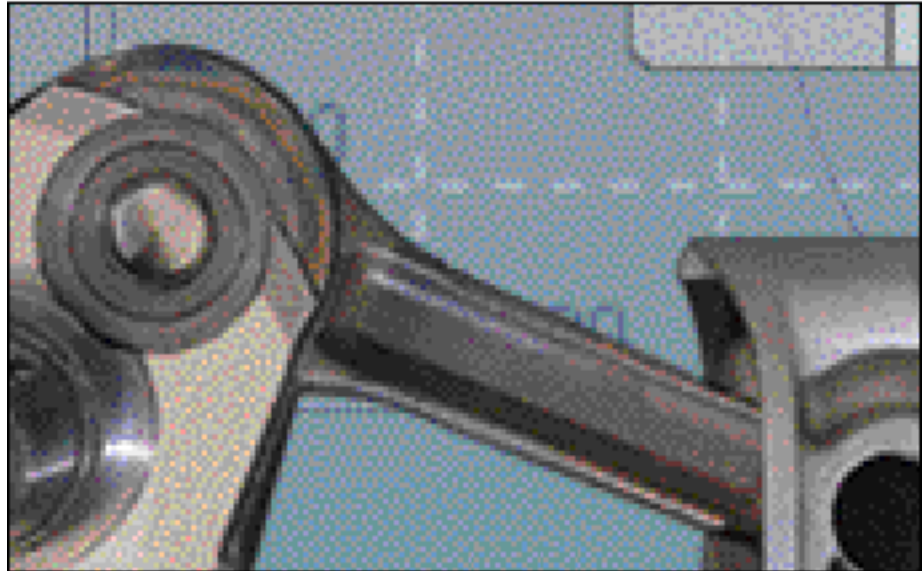


Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Knowledge Revolution.

© 1995 Knowledge Revolution. All rights reserved.
Portions © 1992-1995 Summit Software Company.

Working Model is a registered trademark of Knowledge Revolution. Working Model Basic and WM Basic are trademarks of Knowledge Revolution. All other trademarks are the property of their respective holders.

Contents



Introduction.....	1
Using the WM Basic Language Reference.....	1
Tables.....	2
Language Elements By Category.....	2
Chapter 1: Getting Started with Working Model Basic.....	17
What is Working Model Basic?.....	18
WM Basic Objects.....	29
Putting it All Together: A Simple WM Basic Program.....	37
Where to Go from Here	39
Chapter 2: A–Z Reference	40
Chapter 3: Working Model Basic Extensions Reference.....	517
Chapter 4: Editing and Debugging Scripts.....	641
Script Editor Basics.....	642
Editing Your Scripts	649
Running Your Scripts.....	658
Debugging Your Scripts.....	659
Exiting from Script Editor.....	667
Menu Reference	667
Chapter 5: Editing Custom Dialog Boxes (Windows Only)	673
Overview.....	674
Using the Dialog Editor.....	674
Creating a Custom Dialog Box	678
Editing a Custom Dialog Box.....	683

Editing an Existing Dialog Box.....	695
Testing an Edited Dialog Box.....	697
Incorporating a Dialog Box Template into Your Script.....	699
Exiting from Dialog Editor.....	700
Menu/Tools Reference.....	700
Using a Custom Dialog.....	706
Using a Dynamic Dialog Box.....	709
 Chapter 6: Controlling Working Model from Another	
Application.....	713
Sending a WM Basic Program via DDE.....	714
Sending a WMBasic Program via Apple Events.....	716
 Appendix A: Runtime Error Messages719	
Visual Basic Compatible Error Messages	719
WM Basic-Specific Error Messages	722
Error Messages in Working Model Operations.....	724
 Appendix B: Compiler Error Messages725	
 Appendix C: Language Elements by Platform731	
 Appendix D: WM Basic Limitations747	
 Appendix E: WM Basic/Visual Basic Differences749	
Index.....	757

Introduction

Welcome to the *Working Model Basic User's Manual*. This manual contains both an alphabetic listing of all WM Basic language keywords as well as chapters describing how to use the WM Basic tools such as the Dialog Editor and Script Editor.

Contents

The *Working Model Basic User's Manual* contains the following sections:

- **Chapter 1, Getting Started with Working Model Basic**, provides a brief but comprehensive introduction to this powerful feature.
- **Chapter 2, A-Z Reference**, contains an alphabetical list of all language keywords, including all functions, subroutines, keywords, constants, and so on. This chapter can be used to look up a function's parameters or to understand the purpose of a language element.
- **Chapter 3, Working Model Basic Extensions Reference**, contains an alphabetical list of extensions to the Basic language that are specific to Working Model.
- **Chapter 4, Editing and Debugging Scripts**, explains how to use the Script Editor, a tool that allows you to edit and debug WM Basic scripts.
- **Chapter 5, Editing Custom Dialog Boxes**, explains how to use the Dialog Editor, a tool that enables you to create and modify custom dialog boxes for use within your WM Basic scripts.
- **Chapter 6, Controlling Working Model from Another Application**, describes how to use Working Model as a DDE or Apple events server.
- **Appendix A, Runtime Error Messages**, provides a complete listing of WM Basic runtime error messages, including the error numbers and error text.
- **Appendix B, Compiler Error Messages**, provides a complete listing of WM Basic compiler error messages, including the error numbers and error text.
- **Appendix C, Language Elements by Platform**, provides a listing of all WM Basic language keywords and on which platforms those keywords are supported.
- **Appendix D, WM Basic Limitations**, describes some limits of the WM Basic runtime and compiler.
- **Appendix E, WM Basic/Visual Basic Differences**, describes the differences between WM Basic and Visual Basic 3.0 and Visual Basic for Applications 1.0.

Using the WM Basic Language Reference

The Reference section is organized like a dictionary containing an entry for each language element. In the Reference section, the language elements are categorized as follows:

Category	Description
data type	Any of the supported data types, such as Integer, String, and so on.
function	Language element that takes zero or more parameters, performs an action, and returns a value.

statement	Language element that takes zero or more parameters and performs an action.
operator	Language elements that cause an evaluation to be performed either on one or two operands.
topic	Describes information about a topic rather than a language element.
keyword	Language element that does not fit into any of the other categories.

Each entry in the Reference section contains the following headings:

Heading	Description
Syntax	The syntax of the language element. The conventions used in describing the syntax are described in Chapter 1.
Description	Contains a one-line description of that language element.
Comments	Contains any other important information about that language keyword.
Example	Contains an example of that language keyword in use. The language keyword to which the example applies always appears in bold within the example. An example is provided for every language keyword.
See Also	Contains a list of other entries in the Reference section that relate either directly or indirectly to that language element.
Platforms	Contains a list of platforms that support that language keyword. The word All indicates that the language keyword is supported by all platforms on which WM Basic runs.
Platform Notes	Contains information about that language keyword specific to a given platform.

Tables

Included in the *Working Model Basic User's Manual* are a number of tables used for cross-referencing the language elements. The following tables are included:

- Language Elements by Category. This table lists the language elements by category, providing a one line description for each one. This table appears later in this section.
- Language Elements by Platform. This table lists all WM Basic language elements and on which platforms these elements are supported. This table appears in Appendix C.

Language Elements By Category

The following subsections list WM Basic language elements by category.

Arrays

ArrayDims	Return the number of dimensions of an array
ArraySort	Sort an array
Erase	Erase the elements in one or more arrays
LBound	Return the lower bound of a given array dimension
Option Base	Change the default lower bound for array

	declarations
ReDim	Re-establish the dimensions of an array
UBound	Return the upper bound of a dimension of an array

Clipboard

Clipboard\$ (function)	Return the content of the clipboard as a string
Clipboard\$ (statement)	Set the content of the clipboard
Clipboard.Clear	Clear the clipboard
Clipboard.GetFormat	Get the type of data stored in the clipboard
Clipboard.GetText	Get text from the clipboard
Clipboard.SetText	Set the content of the clipboard to text

Comments

'	Comment to end-of-line
REM	Add a comment

Comparison operators

<	Less than
<=	Less than or equal to
<>	Not equal
=	Equal
>	Greater than
>=	Greater than or equal to

Controlling other programs

AppActivate	Activate an application
AppClose	Close an application
AppFileName\$	Return the filename corresponding to an application
AppFind	Return the full name of an application
AppGetActive\$	Return the name of the active application
AppGetPosition	Get the position and size of an application
AppGetState	Get the window state of an application
AppHide	Hide an application
AppList	Fill an array with a list of running applications
AppMaximize	Maximize an application
AppMinimize	Minimize an application
AppMove	Move an application
AppRestore	Restore an application
AppSetState	Set the state of an application's window
AppShow	Show an application
AppSize	Change the size of an application
AppType	Return the type of an application
DoKeys	Simulate keystrokes in another application
HLine	Scroll the active window left/right by a specified number of lines
HPage	Scroll the active window left/right by a specified number of pages
HScroll	Scroll the active window left/right to a

	specified absolute position
HWND.Value	Return the operating system value of a window
Menu	Execute a menu command in another application
MenuItemChecked	Determine if a menu item is checked in another application
MenuItemEnabled	Determine if a menu item is enabled in another application
MenuItemExists	Determine if a menu item exists in another application
QueEmpty	Empty a queue
QueFlush	Play back all events stored in a queue
QueKeyDn	Add key down event to the queue
QueKeys	Add key down/up events to the queue
QueKeyUp	Add key up event to the queue
QueMouseClicked	Add mouse click to the queue
QueMouseDblClk	Add mouse double-click to the queue
QueMouseDblDn	Add mouse down/up/down events to the queue
QueMouseDown	Add mouse down event to the queue
QueMouseMove	Add mouse move event to the queue
QueMouseMoveBatch	Add many mouse move events to the queue
QueMouseUp	Add mouse up event to the queue
QueSetRelativeWindow	Make all mouse positions in a queue relative to a window
SendKeys	Send keystrokes to another application
Shell	Execute another application
VLine	Scroll the active window up/down by a specified number of lines
VPage	Scroll the active window up/down by a specified number of pages
VScroll	Scroll the active window up/down to a specified absolute position
WinActivate	Activate a window
WinClose	Close a window
WinFind	Find a window given its name
WinList	Fill an array with window objects, one for each top-level window
WinMaximize	Maximize a window
WinMinimize	Minimize a window
WinMove	Move a window
WinRestore	Restore a window
WinSize	Size a window

Controlling program flow

Call	Call a subroutine
Choose	Return a value at a given index
Do...Loop	Execute a group of statements repeatedly
DoEvents (function)	Yield control to other applications

DoEvents (statement)	Yield control to other applications
End	Stop execution of a script
Exit Do	Exit a Do loop
Exit For	Exit a For loop
For...Next	Repeat a block of statement a specified number of times
GoSub	Execute at a specific label, allowing control to return later
Goto	Execute at a specific label
If...Then...Else	Conditionally execute one or more statements
IIf	Return one of two values depending on a condition
Main	Define a subroutine where execution begins
Return	Continue execution after the most recent GoSub
Select...Case	Execute one of a series of statements
Sleep	Pause for a specified number of milliseconds
Stop	Suspend execution, returning to a debugger (if present)
Switch	Return one of a series of expressions depending on a condition
While...Wend	Repeat a group of statements while a condition is True

Controlling the operating environment

Command, Command\$	Return the command line
Desktop.ArrangeIcons	Arrange the icons on the desktop
Desktop.Cascade	Cascades all non-minimized applications
Desktop.SetColors	Set the desktop colors
Desktop.SetWallpaper	Set the desktop wallpaper
Desktop.Snapshot	Capture an image, placing it in the clipboard
Desktop.Tile	Tiles all non-minimized applications
Environm Environ\$	Return a string from the environment
PrinterGetOrientation	Retrieve the current printer orientation
PrinterSetOrientation	Set the printer orientation
Screen.DlgBaseUnitsX	Return the x dialog base units
Screen.DlgBaseUnitsY	Return the y dialog base units
Screen.Height	Return the height of the display, in pixels
Screen.TwipsPerPixelX	Return the number of twips per pixel in the x direction
Screen.TwipsPerPixelY	Return the number of twips per pixel in the y direction
Screen.Width	Return the width of the display, in pixels
System.Exit	Exit the operating environment
System.FreeMemory	Return the free memory in the operating environment
System.FreeResources	Return the free resources in the operating environment
System.MouseTrails	Toggle mouse trails on or off

System.Restart	Restart the operating environment
System.TotalMemory	Return the total available memory in the operating environment
System.WindowsDirectory\$	Return the directory containing Windows
System.WindowsVersion\$	Return the Windows version

Conversion

Asc	Return the value of a character
CBool	Convert a value to a Boolean
CCur	Convert a value to Currency
CDate	Convert a value to a Date
Cdbl	Convert a value to a Double
Chr, Chr\$	Convert a character value to a string
CInt	Convert a value to an Integer
CLng	Convert a value to a Long
CSng	Convert a value to a Single
CStr	Convert a value to a String
CVar	Convert a value to a Variant
CVDate	Convert a value to a Date
CVErr	Convert a value to an error
Hex, Hex\$	Convert a number to a hexadecimal string
IsDate	Determine if an expression is convertible to a date
IsError	Determine if a variant contains a user-defined error value
IsNumeric	Determine if an expression is convertible to a number
Oct, Oct\$	Convert a number to an octal string
Str, Str\$	Convert a number to a string
Val	Convert a string to a number

Data types

Boolean	Data type representing True or False values
Currency	Data type used to hold monetary values
Date	Data type used to hold dates and times
Double	Data type used to hold real number with 15-16 digits of precision
HWND	Data type used to hold windows
Integer	Data type used to hold whole numbers with 4 digits of precision
Long	Data type used to hold whole numbers with 10 digits of precision
Object	Data type used to hold OLE automation objects
Single	Data type used to hold real number with 7 digits of precision
String	Data type used to hold sequences of characters

Variant Data type that holds a number, string, or OLE automation object

Database

SQLBind	Specify where to place results with SQLRetrieve
SQLClose	Close a connection to a database
SQLError	Return error information when an SQL function fails
SQLExecQuery	Execute a query on a database
SQLGetSchema	Return information about the structure of a database
SQLOpen	Establishes a connection with a database
SQLRequest	Run a query on a database
SQLRetrieve	Retrieve all or part of a query
SQLRetrieveToFile	Retrieve all or part of a query, placing results in a file

Date/time

Date, Date\$ (functions)	Return the current date
Date, Date\$ (statements)	Change the system date
DateAdd	Add a number of date intervals to a date
DateDiff	Subtract a number of date intervals from a date
DatePart	Return a portion of a date
DateSerial	Assemble a date from date parts
DateValue	Convert a string to a date
Day	Return the day component of a date value
Hour	Return the hour part of a date value
Minute	Return the minute part of a date value
Month	Return the month part of a date value
Now	Return the date and time
Second	Return the seconds part of a date value
Time, Time\$ (functions)	Return the current system time
Time, Time\$ (statements)	Set the system time
Timer	Return the number of elapsed seconds since midnight
TimeSerial	Assemble a date/time value from time components
TimeValue	Convert a string to a date/time value
Weekday	Return the day of the week of a date value
Year	Return the year part of a date value

DDE

DDEExecute	Execute a command in another application
DDEInitiate	Initiate a DDE conversation with another application
DDEPoke	Set a value in another application
DDERequest, DDERequest\$	Return a value from another application
DDESend	Establishes a DDE conversation, then sets a value in another application

DDETerminate	Terminate a conversation with another application
DDETerminateAll	Terminate all conversations
DDETimeOut	Set the timeout used for non-responding applications

Dialog manipulation

ActivateControl	Activate a control
ButtonEnabled	Determine if a button in another application's dialog is enabled
ButtonExists	Determine if a button in another application's dialog exists
CheckBoxEnabled	Determine if a check box in another application's dialog is enabled
CheckBoxExists	Determine if a check box in another application's dialog exists
ComboBoxEnabled	Determine if a combo box in another application's dialog is enabled
ComboBoxExists	Determine if a combo box in another application's dialog exists
EditEnabled	Determine if a text box in another application's dialog is enabled
EditExists	Determine if a text box in another application's dialog exists
GetCheckBox	Get the state of a check box in another application's dialog box
GetComboBoxItem\$	Get an item from a combo box item in another application's dialog box
GetComboBoxItemCount	Get the number of items in a combo box on another application's dialog box
GetEditText\$	Get the content of a text box in another application's dialog box
GetListBoxItem\$	Get an item from a list box in another application's dialog box
GetListBoxItemCount	Get the number of items from a list box in another application's dialog box
GetOption	Get the state of an option button in another application's dialog box
ListBoxEnabled	Determine if a list box in another application's dialog box is enabled
ListBoxExists	Determine if a list box in another application's dialog box exists
OptionEnabled	Determine if an option button in another application's dialog is enabled
OptionExists	Determine if an option button in another application's dialog exists
SelectButton	Select a push button in another application's dialog box
SelectComboboxItem	Select an item from a combo box in another application's dialog box
SelectListboxItem	Select an item from a list box in another application's dialog box

SetCheckbox	Set the state of a check box in another application's dialog box
SetEditText	Set the content of a text box in another application's dialog box
SetOption	Set the state of an option button in another application's dialog box

Error handling

Erl	Return the line with the error
Err (function)	Return the error that caused the current error trap
Err (statement)	Set the value of the error
Error	Simulate a trappable runtime error
Error, Error\$	Return the text of a given error
On Error	Trap an error
Resume	Continue execution after an error trap

File I/O

Close	Close one or more files
Eof	Determine if the end-of-file has been reached
FreeFile	Return the next available file number
Get	Read data from a random or binary file
Input#	Read data from a sequential file into variables
Input, Input\$	Read a specified number of bytes from a file
Line Input #	Read a line of text from a sequential file
Loc	Return the record position of the file pointer within a file
Lock	Lock a section of a file
Lof	Return the number of bytes in an open file
Open	Open a file for reading or writing
Print #	Print data to a file
Put	Write data to a binary or random file
Reset	Close all open files
Seek	Return the byte position of the file pointer within a file
Seek	Set the byte position of the file pointer which a file
UnLock	Unlock part of a file
Width#	Specify the line width for sequential files
Write #	Write data to a sequential file

File system

ChDir	Change the current directory
ChDrive	Change the current drive
CurDir, CurDir\$	Return the current directory
Dir, Dir\$	Return files in a directory
DiskDrives	Fill an array with valid disk drive letters
DiskFree	Return the free space on a given disk drive
FileAttr	Return the mode in which a file is open

FileCopy	Copy a file
FileDateTime	Return the date and time when a file was last modified
FileDirs	Fill an array with a subdirectory list
FileExists	Determine if a file exists
FileLen	Return the length of a file in bytes
FileList	Fill an array with a list of files
FileParse\$	Return a portion of a filename
FileType	Return the type of a file
GetAttr	Return the attributes of a file
Kill	Delete files from disk
MacID	Return a value representing a collection of same-type files on the Macintosh
MkDir	Create a subdirectory
Name	Rename a file
RmDir	Remove a subdirectory
SetAttr	Change the attributes of a file

Financial

DDB	Return depreciation of an asset using double-declining balance method
Fv	Return the future value of an annuity
IPmt	Return the interest payment for a given period of an annuity
IRR	Return the internal rate of return for a series of payments and receipts
MIRR	Return the modified internal rate of return
NPer	Return the number of periods of an annuity
Npv	Return the net present value of an annuity
Pmt	Return the payment for an annuity
PPmt	Return the principal payment for a given period of an annuity
Pv	Return the present value of an annuity
Rate	Return the interest rate for each period of an annuity
Sln	Return the straight-line depreciation of an asset
SYD	Return the Sum of Years' Digits depreciation of an asset

Getting information from WM Basic

Basic.Capability	Return capabilities of the platform
Basic.Eoln\$	Return the end-of-line character for the platform
Basic.FreeMemory	Return the available memory
Basic.HomeDir\$	Return the directory where WM Basic is located
Basic.OS	Return the platform id
Basic.PathSeparator\$	Return the path separator character for the platform

Basic.Version\$	Return the version of WM Basic
-----------------	--------------------------------

INI Files

ReadIni\$	Read a string from an INI file
ReadIniSection	Read all the item names from a given section of an INI file
WriteIni	Write a new value to an INI file

Logical/binary operators

And	Logical or binary conjunction
Eqv	Logical or binary equivalence
Imp	Logical or binary implication
Not	Logical or binary negation
Or	Logical or binary disjunction
Xor	Logical or binary exclusion

Math

Abs	Return the absolute value of a number
Atn	Return the arc tangent of a number
Cos	Return the cosine of an angle
Exp	Return e raised to a given power
Fix	Return the integer part of a number
Int	Return the integer portion of a number
Log	Return the natural logarithm of a number
Random	Return a random number between two values
Randomize	Initialize the random number generator
Rnd	Generate a random number between 0 and 1
Sgn	Return the sign of a number
Sin	Return the sine of an angle
Sqr	Return the square root of a number
Tan	Return the tangent of an angle

Miscellaneous

()	Force parts of an expression to be evaluated before others
_	Line continuation
Beep	Make a sound
Inline	Allow execution or interpretation of a block of text
MacScript	Execute an AppleScript script
Mci	Execute an MCI command
PrintFile	Print a file using the application to which the file belongs

Networks

Net.AddCon\$	Redirect a local device to a shared device on a network
Net.Browse\$	Display a dialog requesting a network directory or printer resource
Net.CancelCon	Cancel a network connection

Net.Dialog	Display a dialog allowing configuration of the network
Net.GetCaps	Return information about the capabilities of the network
Net.GetCon\$	Return the name of the network resource associated with a local device
Net.User\$	Return the name of the user on the network

Numeric operators

*	Multiply
+	Add
-	Subtract
/	Divide
\	Integer divide
^	Power
Mod	Remainder

Objects

CreateObject	Instantiate an OLE automation object
GetObject	Return an OLE automation object from a file, or returns a previously instantiated OLE automation object
Is	Compare two object variables
Nothing	Value indicating no valid object

Parsing

Item\$	Return a range of items from a string
ItemCount	Return the number of items in a string
Line\$	Retrieve a line from a string
LineCount	Return the number of lines in a string
Word\$	Return a sequence of words from a string
WordCount	Return the number of words in a string

Predefined dialogs

AnswerBox	Display a dialog asking a question
AskBox\$	Display a dialog allowing the user to type a response
AskPassword\$	Display a dialog allowing the user to type a password
InputBox, InputBox\$	Display a dialog allowing the user to type a response
MsgBox (function)	Display a dialog containing a message and some buttons
MsgBox (statement)	Display a dialog containing a message and some buttons
MsgClose	Close a modeless message box
MsgOpen	Open a modeless message box
MsgSetText	Set the message contained within a modeless message box
MsgSetThermometer	Set the percentage of the thermometer in a modeless message box

OpenFilename\$	Display a dialog requesting a file to open
PopupMenu	Display a popup menu containing items from an array
SaveFilename\$	Display a dialog requesting the name of a new file
SelectBox	Display a dialog allowing selection of an item from an array

Printing

Print	Print data to the screen
Spc	Print a number of spaces within a Print statement
Tab	Used with Print to print spaces up to a column position
ViewportClear	Clear the contents of the viewport
ViewportClose	Close the viewport
ViewportOpen	Open a viewport

Procedures

Declare	Define an external routine or a forward reference
Exit Function	Exit a function
Exit Sub	Exit a subroutine
Function...End	Create a user-defined function
Sub...End	Create a user-defined subroutine

String operators

&	Concatenate two strings
Like	Compare a string against a pattern

Strings

Format, Format\$	Return a string formatted to a given specification
InStr	Return the position of one string within another
LCase, LCase\$	Convert a string to lower case
Left, Left\$	Return the left portion of a string
Len	Return the length of a string or the size of a data item
LSet	Left align a string or user-defined type within another
LTrim, LTrim\$	Remove leading spaces from a string
Mid, Mid\$	Return a substring from a string
Mid, Mid\$	Replace one part of a string with another
Option Compare	Change the default comparison between text and binary
Option CStrings	Allow interpretation of C-style escape sequences in strings
Right, Right\$	Return the right portion of a string
RSet	Right align a string within another
RTrim, RTrim\$	Remove trailing spaces from a string
Space, Space\$	Return a string of spaces

StrComp	Compare two strings
String, String\$	Return a string consisting of a repeated character
Trim, Trim\$	Trim leading and trailing spaces from a string
UCase, UCase\$	Return the upper case of a string

User dialogs

Begin Dialog	Begin definition of a dialog template
CancelButton	Define a Cancel button within a dialog template
CheckBox	Define a combo box in a dialog template
ComboBox	Define a combo box in a dialog template
Dialog (function)	Invoke a user-dialog, returning which button was selected
Dialog (statement)	Invoke a user-dialog
DlgControlId	Return the id of a control in a dynamic dialog
DlgEnable	Determine if a control is enabled in a dynamic dialog
DlgEnable	Enable or disables a control in a dynamic dialog
DlgFocus	Return the control with the focus in a dynamic dialog
DlgFocus	Set focus to a control in a dynamic dialog
DlgListBoxArray	Set the content of a list box or combo box in a dynamic dialog
DlgListBoxArray	Set the content of a list box or combo box in a dynamic dialog
DlgSetPicture	Set the picture of a control in a dynamic dialog
DlgText (statement)	Set the content of a control in a dynamic dialog
DlgText\$ (function)	Return the content of a control in a dynamic dialog
DlgValue (function)	Return the value of a control in a dynamic dialog
DlgValue (statement)	Set the value of a control in a dynamic dialog
DlgVisible (function)	Determine if a control is visible in a dynamic dialog
DlgVisible (statement)	Set the visibility of a control in a dynamic dialog
DropListBox	Define a drop list box in a dialog template
GroupBox	Define a group box in a dialog template
ListBox	Add a list box to a dialog template
OKButton	Add an OK button to a dialog template
OptionButton	Add an option button to a dialog template
OptionGroup	Add an option group to a dialog template
Picture	Add a picture control to a dialog template
PictureButton	Add a picture button to a dialog template

PushButton	Add a push button to a dialog template
Text	Add a text control to a dialog template
TextBox	Add a text box to a dialog template

Variables and constants

=	Assignment
Const	Define a constant
DefBool	Set the default data type to Boolean
DefCur	Set the default data type to Currency
DefDate	Set the default data type to Date
DefDbl	Set the default data type to Double
DefInt	Set the default data type to Integer
DefLng	Set the default data type to Long
DefObj	Set the default data type to Object
DefSng	Set the default data type to Single
DefStr	Set the default data type to String
DefVar	Set the default data type to Variant
Dim	Declare a local variable
Global	Declare variables for sharing between scripts
Let	Assign a value to a variable
Private	Declare variables accessible to all routines in a script
Public	Declare variables accessible to all routines in all scripts
Set	Assign an object variable
Type	Declare a user-defined data type

Variants

IsEmpty	Determine if a variant has been initialized
IsError	Determine if a variant contains a user-defined error
IsMissing	Determine if an optional parameter was specified
IsNull	Determine if a variant contains valid data
IsObject	Determine if an expression contains an object
VarType	Return the type of data stored in a variant

Getting Started with Working Model Basic

This chapter is an introduction to Working Model Basic (WM Basic), a powerful programming language embedded in Working Model. An overview of the language will be provided, followed by simple programming examples using Working Model Basic.

Contents

- What is Working Model Basic?
- Working Model Basic Objects
- Putting it All Together: Simple WM Basic Program

Notes: This documentation, *Working Model Basic User's Manual*, is *not* designed as a guide for those who are new to programming. The documentation assumes that you have some experience in programming and that you have functional knowledge of Microsoft Visual Basic. Although this document serves as the complete reference for the Working Model Basic language, you may find the literature on Visual Basic useful in more advanced programming.

The documentation also assumes that you are somewhat familiar with Working Model. If you have never used Working Model, we strongly recommend that you go through several tutorial exercises in the accompanying *Working Model Tutorial*. The Guided Tour chapter in the *Working Model User's Manual* will also help you become familiar with Working Model.

What is Working Model Basic?

WM Basic (short for Working Model Basic) is an object-oriented programming language based on Visual Basic. Visual Basic is a multi-purpose, industry-standard programming language developed by Microsoft Corporation. WM Basic is a combination of the versatile features available in Visual Basic and special extensions that allow easy access to unique simulation features of Working Model.

As a result, WM Basic has all the following characteristics combined:

- traditional high-level programming language features (e.g., English-like syntax, conditional flow control, subroutines), inherited from the original BASIC programming language
- object-oriented language structure and simple interface design tools (e.g., creating dialog boxes and buttons), inherited from Visual Basic
- a complete set of language extensions specific to Working Model

Using WM Basic, you can write programs to create and run simulations in Working Model. This mode of operation—called *scripting*—expands the capability of Working Model enormously. Below are but a few examples of what you can accomplish with WM Basic.

- Run a four-bar linkage simulation repeatedly. Each time, modify the geometry of the links, plot the trajectory of the coupled link, and export the data to a file. Also, save each simulation with a complete simulation history to a hard disk for future reviews.
- Develop a simulation environment using custom-made dialog boxes. The user only needs to specify a set of parameters through graphical user interface controls (such as buttons and menus), and Working Model will automatically create a model that satisfies the parameters. The user does not need to be familiar with Working Model.
- Develop an analysis program on your own and link it with Working Model. You can write a signal processing program to perform frequency-domain analysis on time-domain simulation data.

Scripting and Interactive Operations

To introduce the idea of scripting, let us show you two ways of creating five rectangles of size 10" (width) by 20" (height) located at (0, 0), (20,0), (40,0), (60,0), and (80,0).

Interactive Operation

One way of accomplishing this task is by using the mouse with click-and-drag operations. You would normally perform the task by using the mouse, the toolbar, and the Properties and Geometry windows.

Scripting Operation

Another, easier way of accomplishing this task is by *scripting* through WM Basic. Shown below is a program, or script, which performs the task (characters following the single quote (') are *comments* and not part of the program):

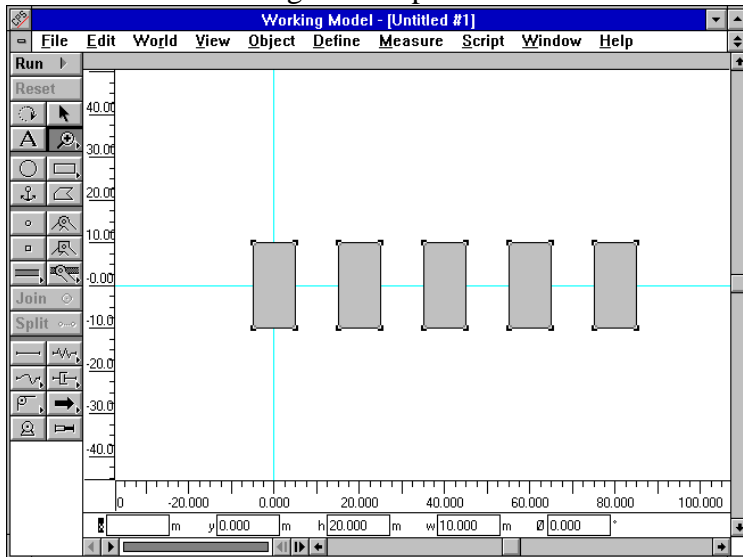
```
Sub Main()
  Dim Rect as WMBody           'Declare the rectangle.
  For I = 0 to 4
    Set Rect = WM.ActiveDocument.NewBody("rectangle")
```

```

Rect.Height.Value = 20
Rect.Width.Value = 10
Rect.PX.Value = I*20    ' position increments by 20
Rect.PY.Value = 0
Next
End Sub

```

The result of executing this script is as follows:



Later, we will show you how to save and run a script.

Advantages of Scripting

Advantages of WM Basic over the interactive use of Working Model include:

- You can pack a frequently-performed series of operations into a *macro*. You can quickly execute these macros as often as you like; scripts can be put under the Script menu of Working Model. You can, of course, share your macros with other users.
- You can combine a set of commands and operations already available in Working Model and write a large program. The new program works as if it were one of the Working Model features, because other users need not understand or be aware of the mechanism of the program.
- You can create a customized simulation/modeling environment using dialog boxes and informative messages. The environment can guide a novice user swiftly through Working Model without fully understanding the application or reading the manuals.
- You can instruct Working Model to perform a series of simulations. Given a carefully written script, Working Model will autonomously repeat simulations while fine-tuning simulation parameters based on the previous results each time, until desired results can be obtained.

Shown above are but few examples of what you can do with scripting. You will realize the unlimited potential using WM Basic as you read through this chapter.

Please note that you are *not* forced to choose either Interactive or Scripting operation. Rather, depending on your application needs, you may wish to operate Working Model completely from scripting level, or you may wish to use scripting as a small macro which accelerates your day-to-day modeling effort. You have enormous flexibility in customizing how you use Working Model.

Objects, Properties, and Methods

This section introduces you to *objects*, *properties*, and *methods*, which are the crucial concepts of WM Basic.

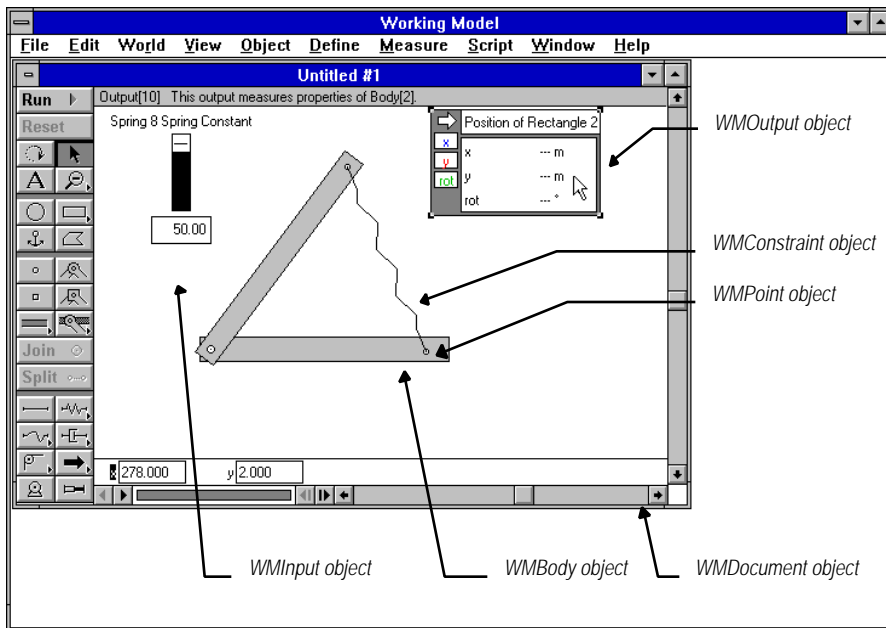
These concepts are common to most object-oriented programming languages, such as C++. If you are experienced in object-oriented programming, you may be already familiar with these terms and concepts.

Objects

Objects are arguably the most basic elements of WM Basic programs. An object is a special data type which holds a set of information (called *properties*) and actions (called *methods*). If you have programmed in another high-level language such as C or PASCAL, you may be familiar with data types called *structures* or *records*. You can think of an object in WM Basic as an abstraction similar to a structure or record, except an object has its own functions and commands (methods) in addition to data members such as integers and strings.

Since an object already has data members and methods built-in, you do not need to define functions and procedures once you define an object in your program. You can immediately start calling the methods associated with the object. For example, once you define a `WMDocument` object, you do not need to define functions or procedures that will create a new body; they are all built-in, as soon as you define the `WMDocument` object.

WM Basic has predefined object types which represent elements of Working Model. Shown below are relationships between objects available in the WM Basic programming language and elements of the Working Model application.



Most objects in WM Basic are direct equivalents of physical objects in Working Model, and they feature very similar, if not identical, functionality. For example, most menu commands are implemented as part of a `WMDocument` object, which represents a Working Model document. Likewise, all the entries in Properties, Geometry, and Appearance windows are available to a `WMBody` object which represents a body in Working Model.

Properties of an Object

A *property* describes a state of the object. A `WMBody` object, for example, has `mass` as one of its properties.

To access the information contained in the properties, you need to specify the object name and add a period (.) followed by the name of the property. For example, if you want to access the mass property of the `WMBody` object called `wheel`, type:

```
Wheel.Mass.Value
```

Most of the properties can be modified as well, for example, in an assignment statement. If you want to change the mass of the `wheel` to 10, simply type:

```
Wheel.Mass.Value = 10
```

(The quantity 10 is interpreted in the current unit system.) Note that the result of this assignment statement is equivalent to that of typing the number "10" in the mass field of the Properties utility window.

Methods of an Object

Besides having properties, objects also know how to perform certain actions. These actions are called *methods*. A `WMDocument` object, for example, has a method called `Run`, which runs the Working Model simulation.

As with properties, if you want to use methods associated with an object, you need to specify the object and add a period (.), followed by the name of the method. For example, suppose

22 Working Model Basic User's Manual

you have a `WMDocument` object called `MyModel`. If you want to run the simulation for 200 frames, type:

```
MyModel.Run 200
```

and the simulation on the document (represented by a `WMDocument` object called `MyModel`) will run for 200 frames, at which point Working Model will automatically pause.

Statements and Functions

Methods can be further subdivided into two categories: *statements* and *functions*. Statements simply execute the given instruction and do not return anything. The `Reset` method shown above is an example of a statement.

On the other hand, functions have a *return value*. For example, the `WMDocument` object has a method called `Body`, which takes the name of a body and looks for an object with a matching name. The `Body` method is a function and therefore has a return value; `Body` either returns the object requested, or returns a special value called `Nothing`, if no object matching the description is found (see “Comparing Objects” on page 26).

For example, you would set the object `Cam1` to a body called “cam lobe” in `Doc1` as follows:

```
Set Cam1 = Doc1.Body("cam lobe")
```

When you browse through Chapters 2 and 3 of this manual for references, you can easily distinguish between statements and functions by looking at the syntax section of each entry. Functions must be used with trailing parentheses at all times, whereas statements must be used *without* parentheses. For example, the syntaxes for the following methods indicate that they are functions (note that a pair of parentheses always follow the keyword, whether or not a parameter is provided):

Syntax `WM.GetMenuItem(Index [, PathString])`

Syntax `WMConstraint.Point(index)`

Syntax `WMDocument.Input(name | id)`

Syntax `WMDocument.NewOutput()`

On the other hand, the syntaxes for the following methods indicate that they are statements.

Syntax `WM.DeleteMenuItem Index`

Syntax `WMConstraint.GetVertex index, x, y`

Syntax `WMDocument.Run [frames]`

Parameters and Return Values for Methods

Most methods take *parameters*, which are values or objects passed as necessary information to invoke the method. The *return value* of a method is yielded as the result of the method. For example, the `NewBody` method (of a `WMDocument` object) takes a `String` parameter which specifies the type of the body to be created. If you typed:

```
Dim MyCircle as WMBody ' declare MyCircle as a WMBody object
Set MyCircle = Doc1.NewBody("circle")
```

then Working Model would create a circle. Note that parameters are provided to functions with a pair of parentheses.

Since the `NewBody` method is a function, it has a return value. In order to invoke any function, you need to store the return value. In the above example, an object of type `MyCircle` is used to store the return value of the `NewBody` method. You cannot simply type:

```
Doc1.NewBody("circle")      ' incorrect usage of a statement.
```

and expect it to be accepted; the return value has to be stored.

On the other hand, when you invoke a statement, you need to specify parameters without using parentheses. For example, when you save a document under a specified name (Save As), type:

```
Doc1.SaveAs "c:\mymodel.wm"
```

on Windows, or

```
Doc1.SaveAs "Macintosh HD:Model 1"
```

on Macintosh.

Another example would be to run a simulation. For instance:

```
Doc1.Run 25
```

will run the simulation for 25 frames and pause.

Note that neither `SaveAs` nor `Run` have a return value since both are statements and not functions.

Declaring Objects

Object types, such as `WMDocument`, define an object and its properties and methods. The type names (e.g., `WMDocument`) are not actual variables but a definition of an object type. To create an object variable, you use the `Dim` statement.¹ For example, to declare `Doc1` as a `WMDocument` object variable, use:

```
Dim Doc1 as WMDocument
```

Likewise, if you want to declare `aBody` as a `WMBody` object variable:

```
Dim aBody as WMBody
```

Now you can use the `Doc1` and `aBody` as `WMDocument` and `WMBody` object variables, respectively.

Creating, Assigning, and Deleting Objects

In most high-level programming languages, you can assign a value to a variable using the equal (=) sign. This rule is true in WM Basic as well. For example:

```
I = 5
```

assigns 5 to the variable `I`. You can use the equal sign to assign variables of types `Integer` (integers), `Double` (floating point numbers) and `String` (character strings).

The only exceptions to this rule are objects. When you want to assign one object to another, you need to use the `Set` statement. For example, the following code segment:

```
Dim BaseMount as WMBody
Set BaseMount = WM.ActiveDocument.NewBody("square")
```

creates a new body (square) in Working Model and assigns the body to the object variable called `BaseMount`. When created with the `NewBody` method, squares have a default dimension (its side is equal to the grid size) and are located at the origin.

The assignment statement using `Set` only assigns *pointers* to actual objects. For example, if the above example is followed by:

¹ `Dim` stands for “dimension” because it needs to allocate a certain space in memory to declare these objects.

24 Working Model Basic User's Manual

```
Dim SubMount as WMBody
Set SubMount = BaseMount
```

then SubMount also “points to” the same square that BaseMount refers to. Therefore, if you modify the property of BaseMount as follows:

```
BaseMount.Width.Value = 1.5
```

then the property SubMount.Width.Value also becomes 1.5.

Likewise, if you delete BaseMount by:

```
WM.ActiveDocument.Delete BaseMount
```

(Delete is a method of a WMDocument object)

then the object SubMount becomes Nothing as well (see “Comparing Objects” below);

accessing properties such as SubMount.Mass results in a run-time error.

Please see the section on Objects in Chapter 2 for more information.

Comparing Objects

When you compare objects in WM Basic, you can use an English-like syntax. For example, the following script searches through the active document for a body called "wheel".

```
Sub Main()
  If WM.ActiveDocument.Body("wheel") is Nothing then
    MsgBox "The wheel cannot be found in the document"
  Else
    MsgBox "There is a wheel"
  End If
End Sub
```

The method Body (of WMDocument objects) returns a special value called Nothing when a specified object could not be found in the document. (Other methods of a WMDocument

object such as Constraint, Point, Input, and Output work the same way; each method looks for a specific type of objects, such as a constraint, point, input, or output.)

The following code segment checks to see if a WMPoint object (called MyPoint) is attached to a WMBody object (called MyBody).² If not, the code will attach MyPoint to MyBody.

```
If MyPoint.Body is not MyBody then
  MsgBox "MyPoint is not attached to MyBody"
  Set MyPoint.Body = MyBody
End If
```

Note the use of the "is not" operator for objects.

Your First WM Basic Program

This section will walk you through creating and running a simple WM Basic program. The program will create a projectile and a meter to measure the position of the projectile.

The sole purpose of this program is to give you an overview of how a WM Basic program might be structured. This program is not designed to make you master the WM Basic programming language—just yet. Do not be alarmed if you cannot understand all the details. If you are experienced in programming in another language, you may wish to try making modifications to the program and test your understanding.

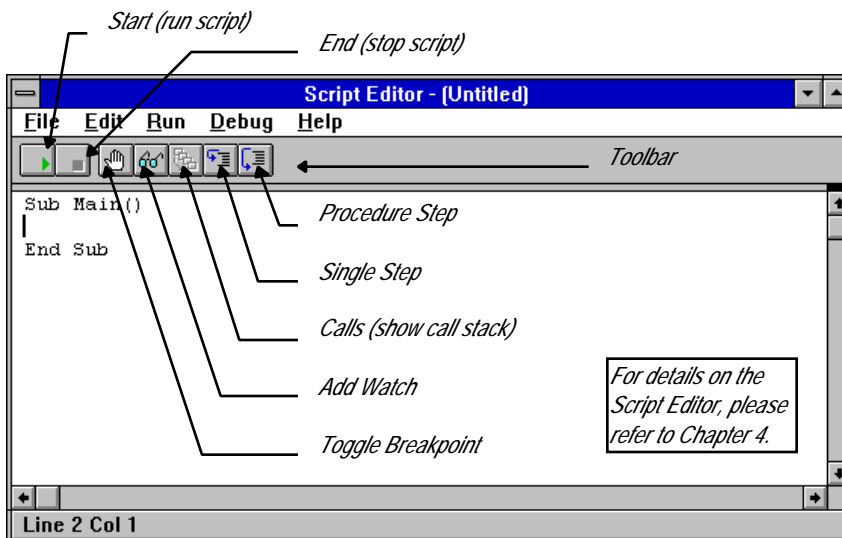
² For more information on WMPoint objects, please refer to the section titled WMPoint (object) in Chapter 3 of this manual.

To Launch the Editor:

You start writing programs by launching the Script Editor within Working Model. (Start Working Model if you have not already done so.)

1. Choose Editor from the Scripts menu.

The Script Editor appears as shown below. The figure shows the Windows version of Script Editor; the Macintosh version is virtually identical.



All WM Basic programs must have at least these two statements as shown in the blank Script Editor window. `Sub Main()` indicates the starting point of the program, so that Working Model knows where to start. `End Sub` indicates that the program ends there. The content of the program is to be "sandwiched" between these two statements.

The Script Editor can stay open while you are running Working Model. At times, Working Model window may cover the Script Editor window. In such cases, either click anywhere in the Script Editor, or choose "Editor" in the Script menu of Working Model; this action will bring the Editor window to the front.

When you no longer need the Script Editor while using Working Model, you can close the window by simply double-clicking on the top left corner of the Script Editor.

To enter the sample program:

1. Start Working Model.
A blank, untitled document appears.
2. Open the Script Editor (see "To Launch the Editor:" above).

Shown below is the code, followed by explanations. A few remarks about WM Basic code should be provided here:

- The single quotation mark (') indicates the beginning of a comment. Any character beyond the comment symbol in each line is ignored by Working Model.

- WM Basic language is *not* case-sensitive. You can use lower- or upper-case letters in the program and they will be interpreted as identical. Therefore, when you define or declare two variables such as `doc` and `Doc`, they will be treated as the same variable.
- You can concatenate two lines of WM Basic using a colon (:).

Enter the following program.

```
Sub Main()                                ' Line 1
  Dim Doc as WMDocument                  ' Line 2
  Dim Sphere as WMBody                    ' Line 3
  Dim Meter as WMOOutput                  ' Line 4
  Set Doc = WM.ActiveDocument             ' Line 5
  Set Sphere = Doc.NewBody("circle")      ' Line 6
  Sphere.PX.Value = 0: Sphere.PY.Value = 0 ' Line 7
  Sphere.Radius.Value = 0.5               ' Line 8
  Set Meter = Doc.NewOutput()             ' Line 9
  Meter.Column(1).Cell.Formula = "body[1].p.y" ' Line 10
  Doc.Run 30                              ' Line 11
End Sub                                   ' Line 12
```

Each line means the following:

- Line 1 serves as the *entry point* (where the program starts) for any WM Basic program. When you start writing programs using functions and procedures, the starting point of a program may not always be located at the first line. The code `Sub Main()` needs to be there to clearly indicate that this line is where the main program starts.
- Lines 2 through 4 are simply declaring object variables to be used for the rest of the program, as Working Model needs to know the types of each object variable before it is used. `WMDocument`, `WMBody`, and `WMOOutput` are all (pre-defined) object type names. From here on, objects `Doc`, `Sphere`, and `Meter` have all the properties and methods associated with each object types. (But you cannot use them just yet, because you need to *assign* objects to these object variables first!)
- Line 5 assigns the object `Doc` to be the current (active) document of the Working Model application. `WM` is a special pre-defined object that represents the Working Model application.
- Line 6 creates a new body (circle) within `Doc`, and assigns the new body to the variable `Sphere`. `NewBody` is a method of any `WMDocument` object. Since the object variable `Doc` has already been defined as a valid `WMDocument` object (in Line 5), `Doc` has access to the method as well.
- Line 7 specifies the position of the circle to be at the origin (0, 0). `PX` and `PY` are properties of `B` (a `WMBody` object). Note how the colon (:) is used to separate two statements on the same line.
- Line 8 specifies the radius of the circle to be 0.5. By default, a circle created by the `NewBody` method has a diameter equal to the current grid size.
- Line 9 creates a new output within `Doc`, and assigns it to the object `Meter`.

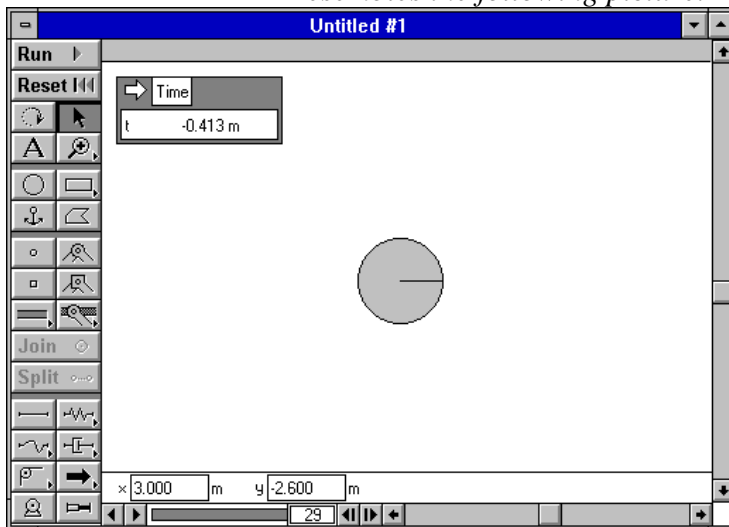
- Line 10 defines the first field of the output `Meter` so that it displays the y position of the body. We typed "`body[1]`" for simplicity, because we know the body created in Line 6 is `body[1]` (this is the first object created in the document).
Note that the columns of a meter need to be specified using Working Model's formula language. The expression "`body[1].p.y`" means the y position of `body[1]`. Please refer to *Working Model User's Manual* for more information.
- Line 11 runs the simulation for 30 frames, after which Working Model will automatically pause.
- Line 12 indicates the exit point of the program. The statement `End Sub` could indicate the end of any program module (such as a subroutine, function, or the main program). In this example, line 12 has the matching entry point `Sub Main()`.

Note how each line of the program uses object methods and creates, assigns, or modifies the object properties.

To run the Sample Program

1. Choose Run under the Run menu of the Script Editor, or simply press the Run button on the Script Editor's toolbar.

Working Model will immediately execute the program. The result resembles the following picture.



To Run the Program Step By Step

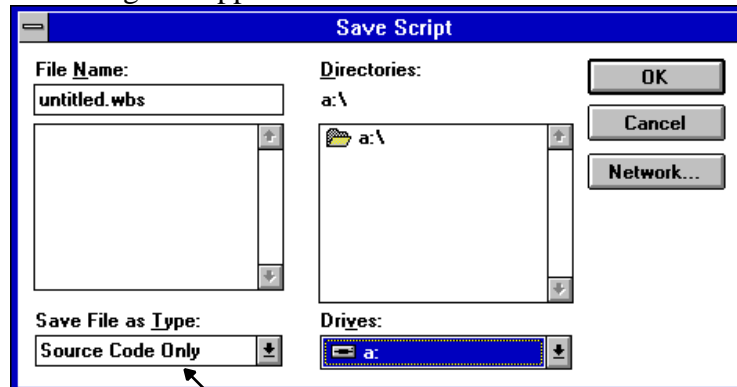
Running a program step by step can be an effective tool for debugging. You can run the program one line at a time and watch each line execute. To run a WM Basic script step by step, simply press the F8 key, or choose Step in the Debug menu of the Script Editor. The source code line that is about to be executed appears highlighted in the Script Editor. For more information on debugging scripts, please see Chapter 4 of this manual.

Saving Your Scripts

You can save the script you have just written to a file.

To Save a Script

1. In the Script Editor, choose Save As in the File menu. The following dialog box appears.



*Choose the desired option (source or object).
On the Macintosh version, a check box appears instead.*

2. On Windows, choose the Source Code Only option. On Macintosh, leave the check box titled "Save script object code only" blank. A script can be saved as source code (*text file*) or as object code (*compiled*). Please see below for implications. On the Windows version, you can choose either option from the pop-up menu. On the Macintosh version, you can click the check box titled "Save script object code only", if you choose to save scripts as object code only.
3. Specify the filename.
4. Click OK.

Saving as Source Code

When a script is saved as text, you are saving exactly what you typed. You can later open, modify, and debug the code. The debugger (please see Chapter 4 for more details) shows the lines of the source code that is being executed.

When a text script is executed, however, Working Model needs to compile the source code before executing it. The compile time is almost negligible for small files, but will increase as the script becomes larger.

By default, a source code text is saved with the .wbs suffix on Windows.

Saving as Object Code

When a script is saved as object code, Working Model will compile the code first and store the result in binary format. You can run the compiled script at a later time, but you will not be able to see or edit the source code. The debugger will not be able to show exact lines of code being executed. If you choose to let other Working Model users run your script but do not want them to see the code, you should distribute the script in the object code format.

WM Basic programs saved as object code are *cross-platform*; you can run them on both Macintosh and Windows Working Model, provided that the program does not use platform-specific features. See Appendix C for the list of platform-specific features.

Since the code is already compiled, a binary script does not require any additional compilation before being executed.

By default, object-only scripts are saved with the `.wbx` suffix on Windows.

WM Basic Objects

Since Working Model handles sophisticated and complex data such as bodies, constraints, and the simulation setup, you need more than integers and strings which are standard data types of a high level programming language. WM Basic has implemented the following object types to make writing simulation scripts simple and easy. This section will show you how WM Basic objects interact with entities in Working Model.

The following objects are discussed in this section:

Object	Description
WMApplication	Represents the Working Model application itself.
WMDocument	Represents a Working Model document.
WMBody	Represents a body in a Working Model document (such as a rectangle, square, circle, and polygon).
WMPoint	Represents a point element in a Working Model document (such as a square point element or an endpoint of a constraint).
WMConstraint	Represents a constraint object in a Working Model document (such as a spring, pin joint, and actuator).
WMOuput	Represents an output object (meter) in a Working Model document.
WMInput	Represents a control object in a Working Model document (such as a slider).

Please refer to Chapter 3, "Working Model Basic Extensions Reference" for complete descriptions on objects, properties, and methods available in WM Basic. The following sections provide brief descriptions of these object types.

WMApplication Object

A `WMApplication` object is the representation of the Working Model application itself.

Below are some examples of what you can accomplish using the `WMApplication` object.

To simplify your work, Working Model has already defined a built-in `WMApplication` object called `WM`. You cannot define any other `WMApplication` object. Please refer to the section on "WM (constant)" in Chapter 3 of this document for its complete description.

Creating a New Document

A WM Basic code to create a new document would be:

```
Dim Doc as WMDocument
Set Doc = WM.New() ' Creates a new document
```

30 Working Model Basic User's Manual

Recall that you need a `Set` statement to assign objects (see “Creating, Assigning, and Deleting Objects” on page 25).

Note that `New` method has a pair of trailing parentheses because `New` is a function (see “Statements and Functions” on page 24).

Choosing and Setting the Active Document

WM Basic provides a facility to access the active document. The *active document* is the document that appears in the foreground of the Working Model application window.

To set a `WMOBJECT` object `Doc` to the currently active Working Model document, type:

```
Set Doc = WM.ActiveDocument
```

When Working Model has multiple documents open, you can control which document should be the active document (document that appears foreground). You simply need to assign the property to the desired document. Suppose Working Model has two documents `Doc1` and `Doc2` open, and you want to set the active document to `Doc2`. You can type:

```
Set WM.ActiveDocument = Doc2 ' brings Doc2 to foreground
```

Opening a Working Model File

To open a file is saved as `"c:\wm30\model1.wm"` on Windows, type:

```
Set Doc = WM.Open("c:\wm30\model1.wm")
```

On Macintosh, use a colon (:) to specify the folder hierarchy. For example, suppose you have a file called `Simulation`, your hard disk is called `Macintosh HD`, and the file is located in Working Model 3.0. Then type:

```
Set Doc = WM.Open("Macintosh HD:Working Model 3.0:Simulation")
```

WMDocument Object

A `WMDocument` object is the representation of a Working Model document. Shown below are some examples of the operations you can accomplish using a `WMDocument` object. Please refer to the section `WMDocument` in Chapter 3 for the complete description.

Saving the Document Under a Specific Name

To save the document `Doc` under a specific name, you would type:

```
Doc.SaveAs "c:\mydir\model1.wm" (on Windows)
```

```
Doc.SaveAs "Hard Disk:My Folder:Model 1" (on Macintosh)
```

Since `SaveAs` is a statement, it has no return value.

You can also store the time history of the simulation. Please refer to the section on `WMDocument.SaveAs` in Chapter 3.

Changing the Simulation Mode

To change the simulation mode, modify the `SimulationMode` property of the `WMDocument` object. For example, if you want to set the current simulation mode of `Doc` to Fast mode (assume `Doc` has been declared and defined as a valid `WMDocument` object):

```
Doc.SimulationMode = "fast" ' set to Fast simulation mode
```

You could also type `"FAST"`, `"Fast"`, or any lower- or upper-case combination. For other simulation modes, please refer to the section `WMDocument.SimulationMode` in Chapter 3.

Changing the Unit System

To change the unit system, simply modify the `UnitSystem` property. For example, if you want to set the current unit system of the document `Doc` (assume it has been declared and defined as a valid `WMDocument` object) to English with pounds:

```
Doc.UnitSystem = "English (earth pounds)"
```

For the complete description, please refer to the section `WMDocument.UnitSystem` in Chapter 3.

Showing and Hiding the Toolbar

You can show or hide the Toolbar by modifying the `ShowToolPalette` property of the `WMDocument` object. The property is of type `Boolean`; you can set it to either `True` or `False` (no quotation marks (") are necessary). By default, `ShowToolPalette` is set to `True`, indicating that the Toolbar is shown. To hide the Toolbar of the document `D` (assume it is declared as a `WMDocument` object):

```
Doc.ShowToolPalette = False      ' Hide the Toolbar
```

Setting it to `True` will show the Toolbar again.

Running and Resetting a Simulation

You can run, pause, and reset the simulation using methods available to `WMDocument` objects. For example, to run a simulation of the document `Doc` (assume it is declared as a `WMDocument` object):

```
Doc.Run      ' runs the simulation indefinitely
```

The simulation runs indefinitely, until stopped by the user through Working Model's graphical interface (such as the Stop button on the toolbar or Pause Control).

If you want to run the simulation for 25 frames and pause, type:

```
Doc.Run 25   ' runs the simulation for 25 frames
```

To reset the simulation:

```
Doc.Reset      ' resets the simulation (has no return code)
```

(Please note that the `Run` method runs the *simulation* but not the *script*. For the `Run` method to take effect, you still need to *execute* the script itself, which can be accomplished by pressing the F5 key.)

WMBody Object

A `WMBody` object is the representation of a physical body in Working Model (such as a circle or a rectangle). The following sections show examples of what you can accomplish using a `WMBody` object. Please refer to the section on `WMBody` in Chapter 3 for the complete description.

Creating a New Body

To create a new body within a document:

1. Declare an object variable of type `WMBody`.
2. Assign the variable by using `NewBody`, which is a method of the `WMDocument` object.

To create a rectangle, for example, type:

```
' Suppose the object Doc is set as a valid WMDocument object.
Dim Rect as WMBody
Set Rect = Doc.NewBody("rectangle")
```

32 Working Model Basic User's Manual

The `NewBody` method takes a string parameter—either "rectangle", "square", "circle", or "polygon"—and creates the body within the document.

Initially, the properties of the body are set to default values. For example, squares and rectangles are initially located at (0, 0), and their width and height are equal to the current grid size. The grid size can be viewed by activating the grid lines (either from the Workspace submenu in the World menu, or by modifying `WMDocument.ShowGridLines` property).

The code shown above will create a rectangle (which looks like a square) at (0, 0). Please proceed to the next section to modify these properties.

Specifying Dimensions and Position of a Body

Once the object is created, you can change its dimensions and position by modifying its properties. For example, suppose you want to set the rectangle just created to be 1 1/2" (width) by 2" (height), located at (1.0, 0). Simply type:

```
Rect.PX.Value = 1.0      ' x-position
Rect.PY.Value = 0        ' y-position
Rect.Width.Value = 1.5   ' width
Rect.Height.Value = 2    ' height
```

Note: the numerical values are interpreted in the current unit system. See “Changing the Unit System” on page 33 for more information.

Note that the properties such as `PX`, `PY`, `Width`, and `Height` are all `WMCell` objects. See the section on `WMCell` in Chapter 3 for more information.

WMPoint Object

`WMPoint` objects represent point elements in Working Model. A `WMPoint` object has properties such as position (local *and* global), and you can modify them. In addition, a `WMPoint` object contains information regarding which body the point is attached to, and which constraint the point is part of. The following sections show examples of using `WMPoint` objects.

Creating a New Point Element

To create a new point element within a document, declare a `WMPoint` object and use `NewPoint`, a method of `WMDocument`.

To create a point element, for example, type:

```
Dim Pt as WMPoint          ' Declare a WMPoint object
Set Pt = WM.ActiveDocument.NewPoint("point")
```

The `NewPoint` method takes a String parameter—"point", "squarepoint", or "anchor"—and creates the corresponding point element within the document. The position of the new point element defaults to (0, 0). Upon creation, the point attaches itself to the background, regardless of whether a body may exist near (0, 0) or not.

Modifying the Position of a Point Element

To modify the position of a point element, simply modify the `PX` and `PY` properties of the `WMPoint` object. For example:

```
' Create a point at (2.0, 3.5).
Dim Pt as WMPoint
Set Pt = WM.ActiveDocument.NewPoint("point")
Pt.PX.Value = 2.0
```

```
Pt.PY.Value = 3.5
```

If you have already attached a point to a body, then the properties `PX` and `PY` hold the local coordinates, given with respect to the frame of reference of the body. You can still access global position of the point as well. See the section “Attaching Points to Bodies” below for more information.

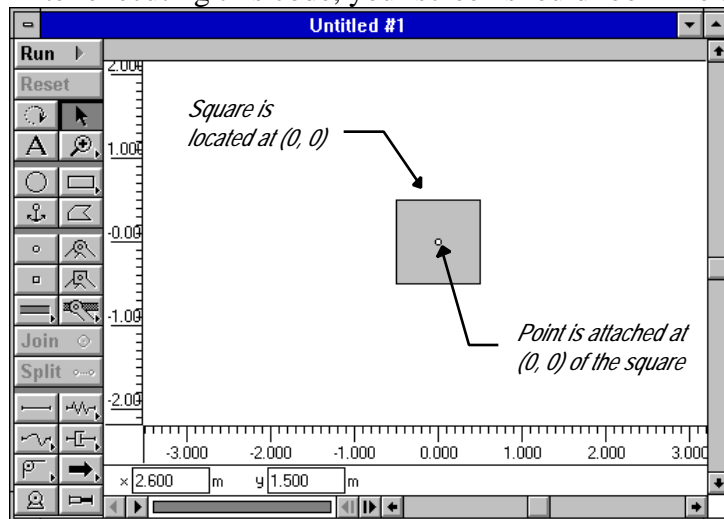
Attaching Points to Bodies

To attach a point to a body, simply assign the `Body` property of the `WMPPoint` object to the desired body. For example, you can create a point and a body (both will default to the position (0, 0) when created), and attach the point to the body in the following fashion:

```
' Create a point and a square. Attach the point to the square.
Dim Pt as WMPPoint
Dim Square as WMBody
Set Square = WM.ActiveDocument.NewBody("square") ' at (0, 0)
Square.Width.Value = 1.0 ' specify the dimension of the square
Set Pt = WM.ActiveDocument.NewPoint("point") ' at (0, 0)
Set Pt.Body = Square ' attaches the point to Square.
Pt.PX.Value = 0.0: Pt.PY.Value = 0.0
```

The last line is not necessary; it is there to emphasize that the position of the point element is now (0, 0) (with respect to the square).

After executing this code, your screen should look like the following:



From then on, the position properties of the `WMPPoint` object will be based on the frame of reference of the `WMBody` object. For example, suppose you added the following code segment to the above example:

```
' Move the square just created to (6.0, 4.0)
B.PX.Value = 6.0
B.PY.Value = 4.0
```

Then the point element `P`, previously attached to the square, would move together with the square. Since the point element is still attached at (0, 0) of the square, the properties `P.PX.Value` and `P.PY.Value` both remain 0.

Accessing Global Position of a Point Element

On the other hand, the global position of this point element is (6.0, 4.0), since the square was moved to that position. To access the global position of the point element, use the expressions `P.GlobalPX` and `P.GlobalPY`.

Note: `GlobalPX` and `GlobalPY` are *not* accessible if formula expressions are used to specify `PX` and `PY` properties of the point element. This is because any formula expressions in `PX` and `PY` override any specifications in `GlobalPX` and `GlobalPY`. For example, if you are using geometry-based expressions (e.g. `body[3].width`, `body[4].radius`) to define the point coordinates on a body, `GlobalPX` and `GlobalPY` are not accessible (i.e., you can neither modify them or read them; they always return 0).

WMConstraint Object

`WMConstraint` objects represent constraints used in Working Model. A `WMConstraint` object has type-dependent properties and information regarding its endpoint(s), and you can modify them. The following sections show examples of what you can accomplish with `WMConstraint` objects.

Creating Constraints

To create a new constraint, declare a `WMConstraint` object and use `NewConstraint`, a method of a `WMDocument` object. You need to specify which type of constraint you wish to create as a parameter to the `NewConstraint` method.

To create a spring, for example, type:

```
Dim Spring as WMConstraint      ' Declare WMconstraint object
Set Spring = WM.ActiveDocument.NewConstraint("spring")
```

The `NewConstraint` method takes a `String` parameter, which specifies the type of the constraint to be created. Once a constraint is created, you *cannot* change its type. Please see the section on `WMConstraint` in Chapter 3 for the full descriptions of the type parameters. The above code segment simply creates a spring, and by default, both endpoints are attached to the background and located at (0, 0). Please see the following sections to attach the endpoints properly to desired bodies and to change properties of the constraint.

Attaching Constraints to Bodies

To attach a constraint to a body,

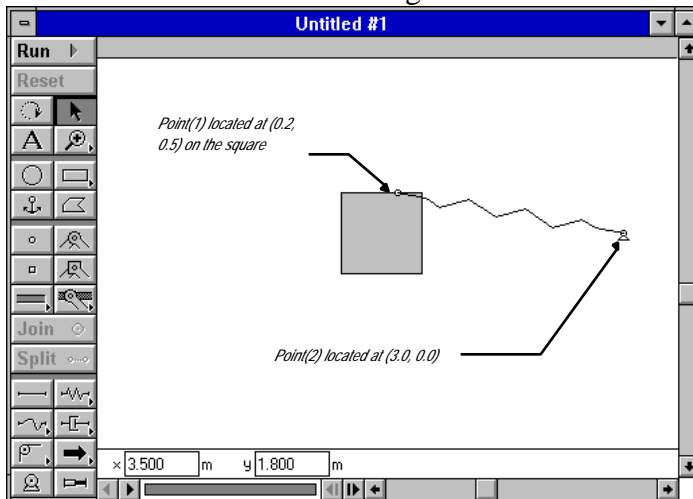
1. Attach the endpoints of the constraint object to the desired body (a `WMBody` object) using the `Point` method of the `WMConstraint`.
2. Modify the positions of the endpoints so that they are attached to the desired location on the body.

For example, suppose `Square` is a body (as created in “`WMBody Object`” on page 33). Then you can create a spring and attach it to the body as follows.

```
' Assume Doc is a valid WMDocument object.
' Assume Square is a valid WMBody object.
Dim Spring as WMConstraint      ' Declare WMconstraint object
Set Spring = Doc.NewConstraint("spring")  ' Create spring
' At this point, both endpoints of the spring are attached
' to the background and located at (0, 0).
' Attach one of the endpoints to B and position the point to
```

```
' (3, 5) on the body.
Set Spring.Point(1).Body = Square
' From here on, the PX/PY fields of Point(1) are measured
' with respect to the FOR of the body.
Spring.Point(1).PX.Value = 0.2
Spring.Point(1).PY.Value = 0.5
' Set the position of the other endpoint to (3, 0) (global)
Spring.Point(2).PX.Value = 3.0
Spring.Point(2).PY.Value = 0.0
```

The result of the above code segment results in the following:



Modifying Constraint Properties

Every constraint has properties that may or may not be unique to its type. For example, a spring has properties such as spring constant and rest length, but these properties are not meaningful for a motor. The section on `WMConstraint` in Chapter 3 describes exactly which properties are available for each type of constraints.

For example, to modify the spring constant to 120, simply type (assume `MySpring` is a valid `WMConstraint` object whose type is `spring`):

```
MySpring.K.Value = 120      ' Change the spring constant to 120
```

To change the natural length of the spring to 2.5 (again, assume `MySpring` is a valid spring constraint):

```
MySpring.Length.Value = 2.5  ' Change the rest length to 2.5
```

Again, please refer to individual sections (such as `WMConstraint.K` and `WMConstraint.Length`) in Chapter 3 for the complete descriptions of these properties.

WMOutput Object

`WMOutput` objects represent meter objects used in Working Model. A `WMOutput` object has properties describing the meter columns, as well as its own position and size in the document window.

Creating Meters

To create a new meter in a document, declare a `WMOutput` object and use `NewOutput`, a method of `WMDocument`. Please see the section on `WMOutput` for more information.

To create a meter, for example, type:

```
Dim Meter as WMOutput          ' Declare WMOutput object
Set Meter = WM.ActiveDocument.NewOutput()
```

By default, the `NewOutput` method will create a time meter. You can customize the meter to measure desired properties, change the meter type to graph plots, and move it to an arbitrary position on the document as well.

Each `WMOutput` object has `x`, `y` (for `x`- and `y`-positions), `width`, and `Height` (width and height) properties. You can modify these properties to position and size the meter arbitrarily.

Specifying Quantities to be Measured

Just like every meter in Working Model, one `WMOutput` object can measure up to four quantities.

Each measured quantity is specified using the `Column` method of the `WMOutput` object. For example, in order to measure the `y`-position of `body[1]`, you would use the following code segment:

```
Dim Meter as WMOutput          ' Declare WMOutput object
Set Meter = WM.ActiveDocument.NewOutput()
' Modify the column y1 to measure the y position of body[1]
Meter.Column(1).Cell.Formula = "body[1].p.y"
Meter.Column(1).Label = "Y-pos of body[1]" ' optional
```

The parameter "1" passed to the `Column` method accesses the first column of the meter (labeled `y1` in the Properties window of the meter). The property `Cell` returns to a `WMCell` object, hence the subsequent property `Formula` (please see the section on `WMCell` in Chapter 3 for more information). The `Label` property is used as an informative label for the user (appears as the column label on the meter) and therefore optional.

Note: You must specify the columns using Working Model's formula expressions or numeric constants. You cannot apply WM Basic expressions such as `"Body.PY.Value"`, since the meter columns are designed to accept only formula expressions or numeric constants.

For the complete descriptions of `WMOutput` objects, please see the section on `WMOutput` in Chapter 3.

WMInput Object

`WMInput` objects represent controls (input objects) used in Working Model. A `WMInput` object has properties regarding its style (e.g., slider, button, textbox), position, and dimensions on the screen.

Creating Inputs

To create a new input in a document, declare a `WMInput` object and use `NewInput`, a method of `WMDocument`. Inputs default to the slider type.

The following example shows how to create a slider in a document. This code segment corresponds to choosing "Generic Control" in the Define -> Control menu.

```
Dim I as WMInput          ' Declare a WMInput object
Set I = D.NewInput()      ' Assume D is a WMDocument
```

Each `WMInput` object has `x`, `y` (for `x`- and `y`-positions), `width`, and `Height` (width and height), just like a `WMOutput` object. You can modify these properties to position and size the `WMInput` object arbitrarily. Please see the section on `WMInput` for more information.

Specifying Objects to be Controlled

To link `WMInput` to other objects in Working Model, you simply need to assign the input ID to the desired property to be controlled.

For example, the following code segment shows how to create a new control to specify the initial x-velocity of a body called `Circle` (assume `Circle` has been declared and defined as a valid `WMBody` object):

```
Dim Slider as WMInput
Set Slider = WM.ActiveDocument.NewInput()
Circle.VX.Formula = "input["+str$(Slider.ID)+"]"
```

The third line above warrants an explanation. Typically, you would specify an expression such as `"input[5]"`, but the ID number of the Input just created may not be necessarily 5. The `ID` property of the `WMInput` object retains the number, and the number is converted to a string by the `str$` function. Finally, the number is concatenated with `"input["` followed by `"]"` to create a valid Working Model formula expression.

Changing the Input Value

An input control in Working Model is a flexible tool which allows you to change its value while the simulation is running. In interactive operation, the user moves the slider bar to change the input values. In scripting operation, you can simply modify the `Value` property of the `WMInput` object to change the value. The `Value` property holds values of type `Double`. For example, the following line sets the slider value of a `WMInput` object called `Slider` to be 5.5.

```
Slider.Value = 5.5
```

Putting it All Together: A Simple WM Basic Program

Now that you know the fundamental elements of Working Model Basic, we will walk you through a simple WM Basic program.

Spring-Mass Simulation

The following code performs the following tasks:

1. Creates a new Working Model document.
2. Creates a disk.
3. Attaches a spring to the disk.
4. Creates an input slider to specify the downward initial velocity of the disk.
4. Creates a meter to measure the y position and velocity of the disk.
5. Runs the simulation for 50 frames and exports the meter data to a file.
6. Closes and saves the document.

Most of the functions are derived from examples provided in the preceding sections in the current chapter.

```
Sub Main()
  ' Declare objects
  Dim Doc as WMDocument
  Dim Disk as WMBody
  Dim Spring1 as WMConstraint
```

```
Dim Slider as WMInput
Dim Meter1 as WMOOutput

' Create a new document
Set Doc = WM.New()

' Change the unit system and view size
Doc.UnitSystem = "si degrees"
Doc.ViewWidth = 20.0
Doc.ScrollTo 0, 0

' Create a circle
Set Disk = Doc.NewBody("circle")
Disk.Radius.Value = 2.0

' Create a spring and attach it to the body
Set Spring1 = Doc.NewConstraint("spring") ' Create spring
Set Spring1.Point(1).Body = Disk         ' Attach to Disk
Spring1.Point(1).PX.Value = 0.0
Spring1.Point(1).PY.Value = 0.0
Spring1.Point(2).PX.Value = 0.0
Spring1.Point(2).PY.Value = 5.0

' Create a new meter
Set Meter1 = Doc.NewOutput()
Dim DiskID as Integer
DiskID = Disk.ID
Meter1.Column(1).Cell.Formula =
"body["+str$(DiskID)+"].p.y"
Meter1.Column(1).Label = "Py"
Meter1.Column(2).Cell.Formula =
"body["+str$(DiskID)+"].v.y"
Meter1.Column(2).Label = "Vy"
Meter1.Column(2).AutoScale = True
Meter1.Format = "Graph"
Meter1.X = 10: Meter1.Y = 10

' Create a new input
Set Slider = Doc.NewInput()
Slider.X = 50: Slider.Y = 150
Disk.VY.Formula = "input["+str$(Slider.ID)+"]"
Slider.Min = -20: Slider.Max = 20 ' set input range
Slider.Value = -5                 ' set current value
Slider.Name = "Initial Y-velocity"

' Run the simulation for 100 frames and reset
Doc.Run 100
Doc.Reset

' Exports the meter data to a file
Doc.ExportStartFrame = 0
Doc.ExportStopFrame = 100
Doc.ExportMeterData "data1.txt"
Doc.Reset

' Saves the simulation file with history
```

```
Doc.SaveAs "pendulum.wm", True  
Doc.Close  
End Sub
```

Where to Go from Here

Now that you know the fundamentals of WM Basic, you are ready to take full advantage of the scripting feature.

If you are interested in more about the Script Editor and its flexible debugging feature. You should consult Chapter 4, Editing and Debugging Scripts. As you write longer and longer scripts, you will find the debugging feature tremendously useful.

Chapter 2 and Chapter 3 provide complete references for all the keywords, functions, methods, and objects available in WM Basic. The chapters show not only syntax and usage but also small examples.

To tie in dialog boxes and other graphical user interface tools of WM Basic, consult Chapter 5 for details. You can create a whole new simulation environment using custom dialog boxes and scripts.

Chapter 6 provides an overview for those interested in using Working Model as a DDE or Apple event server. Other applications such as Excel or MATLAB can control Working Model by sending commands written in WM Basic.

CHAPTER 2

A–Z Reference

This chapter of the *Working Model Basic User's Manual* contains a complete, alphabetical listing of all keywords in the Working Model Basic (WM Basic) language. When syntax is described, the following notations are used:

Notation	Description
<code>While...Wend</code>	Elements belonging to the WM Basic language, referred to in this manual as keywords, appear in the typeface shown to the left.
<i>variable</i>	Items that are to be replaced with information that you supply appear in italics. The type of replacement is indicated in the following description.
<code>text\$</code>	The presence of a type-declaration character following a parameter signifies that the parameter must be a variable of that type or an expression that evaluates to that type.
<code>[parameter]</code>	If a parameter does not appear with a type-declaration character, then its type is described in the text. Square brackets indicate that the enclosed items are optional.
<hr/> Note: In WM Basic, you cannot end a statement with a comma, even if the parameters are optional: <code>MsgBox "Hello",, "Message" 'OK</code> <code>MsgBox "Hello",, 'Not valid</code> <hr/>	
<code>{Input Binary}</code>	Braces indicate that you must choose one of the enclosed items, which are separated by a vertical bar.
<code>...</code>	Ellipses indicate that the preceeding expression can be repeated any number of times.

& (operator)

Syntax *expression1 & expression2*

Description Returns the concatenation of *expression1* and *expression2*.

Comments If both expressions are strings, then the type of the result is `String`. Otherwise, the type of the result is a `String` variant.

When nonstring expressions are encountered, each expression is converted to a `String` variant. If both expressions are `Null`, then a `Null` variant is returned. If only one expression is `Null`, then it is treated as a zero-length string. Empty variants are also treated as zero-length strings.

In many instances, the plus (+) operator can be used in place of &. The difference is that + attempts addition when used with at least one numeric expression, whereas & always concatenates.

Example 'This example assigns a concatenated string to variable s\$ and a string to s2\$, then concatenates the two variables and displays the result in a 'dialog box.

```
Sub Main()  
    s$ = "This string" & " is concatenated"  
    s2$ = " with the & operator."  
    MsgBox s$ & s2$  
End Sub
```

See Also + (operator); Operator Precedence (topic).

Platform(s) Windows and Macintosh.

' (keyword)

Syntax ' *text*

Description Causes the compiler to skip all characters between this character and the end of the current line.

Comments This is very useful for commenting your code to make it more readable.

Example Sub Main()
 'This whole line is treated as a comment.
 i\$ = "Strings" 'This is a valid assignment with a comment.
 This line will cause an error (the apostrophe is missing).
End Sub

See Also Rem (statement); Comments (topic).

Platform(s) Windows and Macintosh.

() (keyword)

Syntax 1 ... (*expression*) ...**Syntax 2** ..., (*parameter*) , ...**Description** Forces parts of an expression to be evaluated before others or forces a parameter to be passed by value.**Comments** **Parentheses within Expressions**

Parentheses override the normal precedence order of WM Basic operators, forcing a subexpression to be evaluated before other parts of the expression. For example, the use of parentheses in the following expressions causes different results:

```
i = 1 + 2 * 3      'Assigns 7.
i = (1 + 2) * 3    'Assigns 9.
```

Use of parentheses can make your code easier to read, removing any ambiguity in complicated expressions.

Parentheses Used in Parameter Passing

Parentheses can also be used when passing parameters to functions or subroutines to force a given parameter to be passed by value, as shown below:

```
ShowForm i          'Pass i by reference.
ShowForm (i)        'Pass i by value.
```

Enclosing parameters within parentheses can be misleading. For example, the following statement appears to be calling a function called `ShowForm` without assigning the result:

```
ShowForm(i)
```

The above statement actually calls a subroutine called `ShowForm`, passing it the variable `i` by value. It may be clearer to use the `ByVal` keyword in this case, which accomplishes the same thing:

```
ShowForm ByVal i
```

Note: The result of an expression is always passed by value.

Example 'This example uses parentheses to clarify an expression.

```
Sub Main()
    bill = False
    dave = True
    jim = True

    If (dave And bill) Or (jim And bill) Then
        MsgBox "The required parties for the meeting are here."
    Else
        MsgBox "Someone is late again!"
    End If
End Sub
```

See Also ByVal (keyword); Operator Precedence (topic).

Platform(s) Windows and Macintosh.

* (operator)

Syntax *expression1* * *expression2*

Description Returns the product of *expression1* and *expression2*.

Comments The result is the same type as the most precise expression, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
Single	Long	Double
Boolean	Boolean	Integer
Date	Date	Double

When the * operator is used with variants, the following additional rules apply:

- Empty is treated as 0.
- If the type of the result is an Integer variant that overflows, then the result is automatically promoted to a Long variant.
- If the type of the result is a Single, Long, or Date variant that overflows, then the result is automatically promoted to a Double variant.
- If *expression1* is Null and *expression2* is Boolean, then the result is Empty. Otherwise, If either expression is Null, then the result is Null.

44 Working Model Basic User's Manual

Example 'This example assigns values to two variables and their product to 'a third variable, then displays the product of s# * t#.

```
Sub Main()  
    s# = 123.55  
    t# = 2.55  
    u# = s# * t#  
    MsgBox s# & " * " & t# & " = " & s# * t#  
End Sub
```

See Also Operator Precedence (topic).

Platform(s) Windows and Macintosh.

+ (operator)

Syntax *expression1* + *expression2*

Description Adds or concatenates two expressions.

Comments Addition operates differently depending on the type of the two expressions:

If one expression is	and the other expression is	then
Numeric	Numeric	Perform a numeric add (see below).
String	String	Concatenate, returning a string.
Numeric	String	A runtime error is generated.
Variant variant.	String	Concatenate, returning a String
Variant	Numeric	Perform a variant add (see below).
Empty variant	Empty variant	Return an Integer variant, value 0.
Empty variant	Boolean variant	Return an Integer variant (value 0 or - 1)
Empty variant unchanged.	Any data type	Return the non-Empty expression
Null variant	Any data type	Return Null.
Variant	Variant	If either is numeric, add; otherwise, concatenate.

When using + to concatenate two variants, the result depends on the types of each variant at runtime. You can remove any ambiguity by using the & operator.

Numeric Add

A numeric add is performed when both expressions are numeric (i.e., not variant or string). The result is the same type as the most precise expression, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
---------------------------------	--	---

Single	Long	Double
--------	------	--------

Boolean	Boolean	Integer
---------	---------	---------

A runtime error is generated if the result overflows its legal range.

Variant Add

If both expressions are variants, or one expression is numeric and the other expression is Variant, then a variant add is performed. The rules for variant add are the same as those for normal numeric add, with the following exceptions:

- If the type of the result is an Integer variant that overflows, then the result is a Long variant.
- If the type of the result is a Long, Single, or Date variant that overflows, then the result is a Double variant.

Example

'This example assigns string and numeric variable values and then uses the + operator to concatenate the strings and form the sums of numeric variables.

```
Sub Main()
    i$ = "Concatenation" + " is fun!"
    j% = 120 + 5          'Addition of numeric literals
    k# = j% + 2.7         'Addition of numeric variable
    MsgBox "This concatenation becomes: '" i$ + Str(j%) + Str(k#) & "'"
End Sub
```

See Also & (operator); Operator Precedence (topic).

Platform(s) Windows and Macintosh.

– (operator)

Syntax 1 *expression1 – expression2*

Syntax 2 *–expression*

Description Returns the difference between *expression1* and *expression2* or, in the second syntax, returns the negation of *expression*.

Comments Syntax 1

The type of the result is the same as that of the most precise expression, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
---------------------------------	--	---

Long	Single	Double
------	--------	--------

Boolean	Boolean	Integer
---------	---------	---------

A runtime error is generated if the result overflows its legal range.

When either or both expressions are *Variant*, then the following additional rules apply:

- If *expression1* is *Null* and *expression2* is *Boolean*, then the result is *Empty*. Otherwise, if either expression is *Null*, then the result is *Null*.
- *Empty* is treated as an *Integer* of value 0.
- If the type of the result is an *Integer* variant that overflows, then the result is a *Long* variant.
- If the type of the result is a *Long*, *Single*, or *Date* variant that overflows, then the result is a *Double* variant.

Syntax 2

If *expression* is numeric, then the type of the result is the same type as *expression*, with the following exception:

- If *expression* is *Boolean*, then the result is *Integer*.

Note: In 2's compliment arithmetic, unary minus may result in an overflow with *Integer* and *Long* variables when the value of *expression* is the largest negative number representable for that data type. For example, the following generates an overflow error:

```
Sub Main()  
  Dim a As Integer  
  a = -32768  
  a = -a           'Generates overflow here.  
End Sub
```

When negating variants, overflow will never occur because the result will be automatically promoted: integers to longs and longs to doubles.

Example 'This example assigns values to two numeric variables and their difference to a third variable, then displays the result.

```
Sub Main()
    i% = 100
    j# = 22.55
    k# = i% - j#
    MsgBox "The difference is: " & k#
End Sub
```

See Also Operator Precedence (topic).

Platform(s) Windows and Macintosh.

. (keyword)

Syntax 1 *object.property*

Syntax 2 *structure.member*

Description Separates an object from a property or a structure from a structure member.

Examples 'This example uses the period to separate an object from a property.

```
Sub Main()
    MsgBox Clipboard.GetText()
End Sub
```

'This example uses the period to separate a structure from a member.

```
Type Rect
    left As Integer
    top As Integer
    right As Integer
    bottom As Integer
End Type
```

```
Sub Main()
    Dim r As Rect
    r.left = 10
    r.right = 12
End Sub
```

See Also Objects (topic).

Platform(s) Windows and Macintosh.

/ (operator)

Syntax *expression1 / expression2*

Description Returns the quotient of *expression1* and *expression2*.

Comments The type of the result is `Double`, with the following exceptions:

If one expression is	and the other expression is	then the type of the result is
---------------------------------	--	---

Integer	Integer	Single
---------	---------	--------

Single	Single	Single
--------	--------	--------

Boolean	Boolean	Single
---------	---------	--------

A runtime error is generated if the result overflows its legal range.

When either or both expressions is `Variant`, then the following additional rules apply:

- If *expression1* is `Null` and *expression2* is `Boolean`, then the result is `Empty`. Otherwise, if either expression is `Null`, then the result is `Null`.
- `Empty` is treated as an `Integer` of value 0.
- If both expressions are either `Integer` or `Single` variants and the result overflows, then the result is automatically promoted to a `Double` variant.

Example 'This example assigns values to two variables and their quotient to a 'third variable, then displays the result.

```
Sub Main()  
    i% = 100  
    j# = 22.55  
    k# = i% / j#  
    MsgBox "The quotient of i/j is: " & k#  
End Sub
```

See Also \ (operator); Operator Precedence (topic).

Platform(s) Windows and Macintosh.

< (operator)

See Comparison Operators (topic).

<= (operator)

See Comparison Operators (topic).

<> (operator)

See Comparison Operators (topic).

= (statement)

Syntax *variable = expression***Description** Assigns the result of an expression to a variable.**Comments** When assigning expressions to variables, internal type conversions are performed automatically between any two numeric quantities. Thus, you can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This occurs when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error:

```

Dim amount As Long
Dim quantity As Integer

amount = 400123           'Assign a value out of range for int.
quantity = amount         'Attempt to assign to Integer.

```

When performing an automatic data conversion, underflow is not an error.

The assignment operator (=) cannot be used to assign objects. Use the `Set` statement instead.

Example

```

Sub Main()
    a$ = "This is a string"
    b% = 100
    c# = 1213.3443
    MsgBox a$ & ", " & b% & ", " & c#
End Sub

```

See Also `Let (statement)`; `Operator Precedence (topic)`; `Set (statement)`; `Expression Evaluation (topic)`.**Platform(s)** Windows and Macintosh.

= (operator)

See `Comparison Operators (topic)`.

> (operator)

See `Comparison Operators (topic)`.

>= (operator)

See `Comparison Operators (topic)`.

\ (operator)

Syntax *expression1 \ expression2*

Description Returns the integer division of *expression1* and *expression2*.

Comments Before the integer division is performed, each expression is converted to the data type of the most precise expression. If the type of the expressions is either Single, Double, Date, or Currency, then each is rounded to Long.

If either expression is a Variant, then the following additional rules apply:

- If either expression is Null, then the result is Null.
- Empty is treated as an Integer of value 0.

Example 'This example assigns the quotient of two literals to a variable and 'displays the result.

```
Sub Main()  
    s% = 100.99 \ 2.6  
    MsgBox "Integer division of 100.99\2.6 is: " & s%  
End Sub
```

See Also / (operator); Operator Precedence (topic).

Platform(s) Windows and Macintosh.

^ (operator)

Syntax *expression1* ^ *expression2*

Description Returns *expression1* raised to the power specified in *expression2*.

Comments The following are special cases:

Special Case	Value
n^0	1
0^{-n}	Undefined
0^{+n}	0
1^n	1

The type of the result is always Double, except with Boolean expressions, in which case the result is Boolean. Fractional and negative exponents are allowed.

If either expression is a Variant containing Null, then the result is Null.

It is important to note that raising a number to a negative exponent produces a fractional result.

Example

```
Sub Main()
    s# = 2 ^ 5           'Returns 2 to the 5th power.
    r# = 16 ^ .5         'Returns the square root of 16.
    MsgBox "2 to the 5th power is: " & s#
    MsgBox "The square root of 16 is: " & r#
End Sub
```

See Also Operator Precedence (topic).

Platform(s) Windows and Macintosh.

_ (keyword)

Syntax `s$ = "This is a very long line that I want to split " + _
"onto two lines"`

Description Line-continuation character, which allows you to split a single WM Basic statement onto more than one line.

Comments The line-continuation character cannot be used within strings and must be preceded by white space (either a space or a tab).

The line-continuation character can be followed by a comment, as shown below:

```
i = 5 + 6 & _      'Continue on the next line.
    "Hello"
```

Example

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    'The line-continuation operator is useful when concatenating
    'long strings.

    msg = "This line is a line of text that" + crlf + "extends beyond "
    _
    + "the borders of the editor" + crlf + "so it is split into
    " _
    + "multiple lines"

    'It is also useful for separating and continuing long calculation
    lines.

    b# = .124
    a# = .223
    s# = ( (((Sin(b#) ^ 2) + (Cos(a#) ^ 2)) ^ .5) / _
        (((Sin(a#) ^ 2) + (Cos(b#) ^ 2)) ^ .5) ) * 2.00
    MsgBox msg & crlf & "The value of s# is: " & s#
End Sub
```

Platform(s) Windows and Macintosh.

Abs (function)

Syntax `Abs (expression)`

Description Returns the absolute value of *expression*.

Comments If *expression* is Null, then Null is returned. Empty is treated as 0.

The type of the result is the same as that of *expression*, with the following exceptions:

- If *expression* is an Integer that overflows its legal range, then the result is returned as a Long. This only occurs with the largest negative Integer:

```
Dim a As Variant
Dim i As Integer
i = -32768
a = Abs(i)           'Result is a Long.
i = Abs(i)           'Overflow!
```

- If *expression* is a Long that overflows its legal range, then the result is returned as a Double. This only occurs with the largest negative Long:

```
Dim a As Variant
Dim l As Long
l = -2147483648
a = Abs(l)           'Result is a Double.
l = Abs(l)           'Overflow!
```

- If *expression* is a Currency value that overflows its legal range, an overflow error is generated.

Example 'This example assigns absolute values to variables of four types and 'displays the result.

```
Sub Main()
    s1% = Abs(- 10.55)
    s2& = Abs(- 10.55)
    s3! = Abs(- 10.55)
    s4# = Abs(- 10.55)
    MsgBox "The absolute values are: " & s1% & ", " & s2& & ", " & s3! &
    ", " & s4#
End Sub
```

See Also Sgn (function).

Platform(s) Windows and Macintosh.

ActivateControl (statement)

Syntax ActivateControl *control*

Description Sets the focus to the control with the specified name or ID.

Comments The *control* parameter specifies either the name or the ID of the control to be activated, as shown in the following table:

If control is	Then
String	<p>A control associated with that name is activated.</p> <p>For push buttons, option buttons, or check boxes, the control with this name is activated. For list boxes, combo boxes, and text boxes, the control that immediately follows the text control with this name is activated.</p>
Numeric	<p>A control with this ID is activated. The ID is first converted to an Integer.</p> <p>The <code>ActivateControl</code> statement generates a runtime error if the dialog control referenced by <i>control</i> cannot be found.</p> <p>You can use the <code>ActivateControl</code> statement to set the focus to a custom control within a dialog box. First, set the focus to the control that immediately precedes the custom control, then simulate a Tab keypress, as in the following example:</p>

```
ActivateControl "Portrait"
DoKeys "{TAB}"
```

Note: The `ActivateControl` statement is used to activate a control in another application's dialog box. Use the `DlgFocus` statement to activate a control in a dynamic dialog box.

Example 'This example runs Notepad using Program Manager's Run command. It uses the `ActivateControl` command to switch focus between the different controls of the Run dialog box.

```
Sub Main()
    If AppFind$("Program Manager") = "" Then Exit Sub
    AppActivate "Program Manager"
    Menu "File.Run"
    SendKeys "Notepad"
    ActivateControl "Run minimized"
    SendKeys " "
    ActivateControl "OK"
    SendKeys "{Enter}"
End Sub
```

See Also `DlgFocus` (statement).

Platform(s) Windows.

And (operator)

Syntax *expression1* And *expression2*

Description Performs a logical or binary conjunction on two expressions.

Comments If both expressions are either Boolean, Boolean variants, or Null variants, then a logical conjunction is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	Null
Null	True	Null
Null	False	False
Null	Null	Null

Binary Conjunction

If the two expressions are Integer, then a binary conjunction is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long, and a binary conjunction is then performed, returning a Long result.

Binary conjunction forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:

1	And	1	=	1	Example:
0	And	1	=	0	5 00001001
1	And	0	=	0	<u>6 00001010</u>
0	And	0	=	0	And 00001000

Example

```

Sub Main()
    n1 = 1001
    n2 = 1000
    b1 = True
    b2 = False
    'This example performs a numeric bitwise And operation and stores
the
    'result in N3.
    n3 = n1 And n2

    'This example performs a logical And comparing B1 and B2 and
displays the
    'result.
    If b1 And b2 Then
        MsgBox "b1 and b2 are True; n3 is: " & n3
    Else
        MsgBox "b1 and b2 are False; n3 is: " & n3
    End If
End Sub

```

See Also Operator Precedence (topic); Or (operator); Xor (operator); Eqv (operator); Imp (operator).

Platform(s) Windows and Macintosh.

AnswerBox (function)

Syntax AnswerBox(prompt [,button1][,button2][,button3][...])

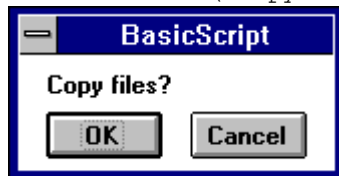
Description Displays a dialog box prompting the user for a response and returns an Integer indicating which button was clicked (1 for the first button, 2 for the second, and so on).

Comments The AnswerBox function takes the following parameters:

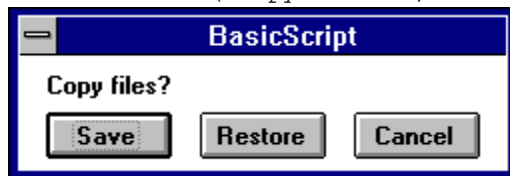
Parameter	Description
<i>prompt</i>	<p>Text to be displayed above the text box. The <i>prompt</i> parameter can be any expression convertible to a String.</p> <p>WM Basic resizes the dialog box to hold the entire contents of <i>prompt</i>, up to a maximum width of 5/8 of the width of the screen and a maximum height of 5/8 of the height of the screen. WM Basic word-wraps any lines too long to fit within the dialog box and truncates all lines beyond the maximum number of lines that fit in the dialog box.</p> <p>You can insert a carriage-return/line-feed character in a string to cause a line break in your message.</p> <p>A runtime error is generated if this parameter is Null.</p>
<i>button1</i>	<p>Text for the first button. If omitted, then "OK" and "Cancel" are used. A runtime error is generated if this parameter is Null.</p>

- button2* Text for the second button. A runtime error is generated if this parameter is Null.
- button3* Text for the third button. A runtime error is generated if this parameter is Null. The width of each button is determined by the width of the widest button. The AnswerBox function returns 0 if the user selects Cancel.

```
r% = AnswerBox("Copy files?")
```



```
r% = AnswerBox("Copy files?", "Save", "Restore", "Cancel")
```



Example 'This example displays a dialog box containing three buttons. It displays
'an additional message based on which of the three buttons is selected.

```
Sub Main()  
    r% = AnswerBox("Copy files?", "Save", "Restore", "Cancel")  
    Select Case r%  
        Case 1  
            MsgBox "Files will be saved."  
        Case 2  
            MsgBox "Files will be restored."  
        Case Else  
            MsgBox "Operation canceled."  
    End Select  
End Sub
```

See Also MsgBox (statement); AskBox\$ (function); AskPassword\$ (function); InputBox, InputBox\$ (functions); OpenFilename\$ (function); SaveFilename\$ (function); SelectBox (function).

Platform(s) Windows and Macintosh.

Platform Notes: AnswerBox displays all text in its dialog box in 8-point MS Sans Serif.

Windows

Any (data type)

Description Used with the `Declare` statement to indicate that type checking is not to be performed with a given argument.

Comments Given the following declaration:

```
Declare Sub Foo Lib "FOO.DLL" (a As Any)

the following calls are valid:
```

```
Foo 10
Foo "Hello, world."
```

Example 'The following example calls the `FindWindow` to determine if Program Manager
'is running. This example will only run under Windows.

'This example uses the `Any` keyword to pass a `NULL` pointer, which is
accepted
'by the `FindWindow` function.

```
Declare Function FindWindow16 Lib "user" Alias "FindWindow" (ByVal
Class _
    As Any,ByVal Title As Any) As Integer

Sub Main()
    Dim hWnd As Variant

    If Basic.Os = vbWin16 Then
        hWnd = FindWindow16("PROGMAN",0&)
    Else
        hWnd = 0
    End If

    If hWnd <> 0 Then
        MsgBox "Program manager is running, window handle is " & hWnd
    End If
End Sub
```

See Also `Declare (statement)`.

Platform(s) Windows and Macintosh.

AppActivate (statement)

Syntax `AppActivate name$ / taskID`

Description Activates an application given its name or task ID.

58 Working Model Basic User's Manual

Comments The `AppActivate` statement takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the application to be activated.
<i>taskID</i>	Number specifying the task ID of the application to be activated. Acceptable task IDs are returned by the <code>Shell</code> function.
Note: When activating applications using the task ID, it is important to declare the variable used to hold the task ID as a <code>Variant</code> . The type of the ID depends on the platform on which WM Basic is running.	

Examples

```
'This example activates Program Manager.

Sub Main()
    AppActivate "Program Manager"
End Sub

'This example runs another application, then activates it.

Sub Main()
    Dim id as variant
    id = Shell("Notepad",7)           'Run Notepad minimized.
    AppActivate "Program Manager"    'Activate Program Manager.
    AppActivate id                   'Now activate Notepad.
End Sub
```

See Also `Shell` (function); `SendKeys` (statement); `WinActivate` (statement).

Platform(s) Windows and Macintosh..

Platform The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

Notes:

Windows Minimized applications are not restored before activation. Thus, activating a minimized DOS application will not restore it; rather, it will highlight its icon.

A runtime error results if the window being activated is not enabled, as is the case if that application is currently displaying a modal dialog box.

Platform On the Macintosh, the *name\$* parameter specifies the title of the desired application. The `MacID` function can be used to specify the application signature of the application to be activated:

```
AppActivate MacID(text$) | task
```

The *text\$* parameter is a four-character string containing an application signature. A runtime error occurs if the `MacID` function is used on platforms other than the Macintosh.

AppClose (statement)

Syntax `AppClose [name$]`

Description Closes the named application.

Comments The *name\$* parameter is a `String` containing the name of the application. If the *name\$* parameter is absent, then the `AppClose` statement closes the active application.

Example 'This example activates Excel, then closes it.

```
Sub Main()
    If AppFind$("Microsoft Excel") = "" Then      'Make sure Excel is
there.
        MsgBox "Excel is not running."
        Exit Sub
    End If
    AppActivate "Microsoft Excel"                  'Activate it (unnecessary).
    AppClose "Microsoft Excel"                      'Close it.
End Sub
```

See Also `AppMaximize (statement)`; `AppMinimize (statement)`; `AppRestore (statement)`; `AppMove (statement)`; `AppSize (statement)`.

Platform(s) Windows.

Platform Notes: A runtime error results if the application being closed is not enabled, as is the case if that application is currently displaying a modal dialog box.

Windows The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppFilename\$ (function)

Syntax `AppFilename$ ([name$])`

Description Returns the filename of the named application.

Comments The *name\$* parameter is a String containing the name of the desired application. If the *name\$* parameter is omitted, then the AppFilename\$ function returns the filename of the active application.

Example

```
'This example switches the focus to Excel, then changes the current
directory
'to be the same as that of Excel.

Sub Main()
    If AppFind$("Microsoft Excel") = "" Then      'Make sure Excel is
there.
        MsgBox "Excel is not running."
        Exit Sub
    End If
    AppActivate "Microsoft Excel" 'Activate Excel.
    s$ = AppFilename$                'Find where the Excel executable is.
    d$ = FileParse$(s$,2)            'Get the path portion of the
filename.
    MsgBox d$                        'Display directory name.
End Sub
```

See Also AppFind\$ (function).

Platform(s) Windows.

Platform For DOS applications launched from Windows, the AppFilename function
Notes: returns the name of the DOS program, not winoldap.exe.

Windows The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppFind\$ (function)

Syntax AppFind\$(*partial_name\$*)

Description Returns a String containing the full name of the application matching the *partial_name\$*.

Comments The *partial_name\$* parameter specifies the title of the application to find. If there is no exact match, WM Basic will find an application whose title begins with *partial_name\$*.

AppFind\$ returns a zero-length string if the specified application cannot be found.

AppFind\$ is generally used to determine whether a given application is running. The following expression returns True if Microsoft Word is running:

```
AppFind$("Microsoft Word")
```

Example 'This example checks to see whether Excel is running before activating it.

```
Sub Main()  
  If AppFind$("Microsoft Excel") <> "" Then  
    AppActivate "Microsoft Excel"  
  Else  
    MsgBox "Excel is not running."  
  End If  
End Sub
```

See Also AppFileName\$ (function).

Platform(s) Windows.

Platform Under Windows, this function returns a String containing the exact text

Notes: appearing in the title bar of the active application's main window.

Windows

AppGetActive\$ (function)

Syntax AppGetActive\$()

Description Returns a String containing the name of the application.

Comments If no application is active, the AppGetActive\$ function returns a zero-length string.

You can use AppGetActive\$ to retrieve the name of the active application. You can then use this name in calls to routines that require an application name.

Example

```
Sub Main()  
  n$ = AppGetActive$()  
  AppMinimize n$  
End Sub
```

See Also AppActivate (statement); WinFind (function).

Platform(s) Windows.

Platform Under Windows, this function returns a `String` containing the exact text

Notes: appearing in the title bar of the active application's main window.

Windows

AppGetPosition (statement)

Syntax `AppGetPosition X,Y,width,height [,name$]`

Description Retrieves the position of the named application.

Comments The `AppGetPosition` statement takes the following parameters:

Parameter	Description
<i>X, Y</i>	Names of <code>Integer</code> variables to receive the position of the application's window.
<i>width, height</i>	Names of <code>Integer</code> variables to receive the size of the application's window.
<i>name\$</i>	<code>String</code> containing the name of the application. If the <i>name\$</i> parameter is omitted, then the active application is used.

The *x*, *y*, *width*, and *height* variables are filled with the position and size of the application's window. If an argument is not a variable, then the argument is ignored, as in the following example, which only retrieves the *x* and *y* parameters and ignores the *width* and *height* parameters:

```
Dim x as integer, y as integer
AppGetPosition x,y,0,0,"Program Manager"
```

Example

```
Sub Main()
  Dim x As Integer, y As Integer
  Dim cx As Integer, cy As Integer
  AppGetPosition x,y,cx,cy,"Program Manager"
End Sub
```

See Also `AppMove (statement)`; `AppSize (statement)`.

Platform(s) Windows.

Platform The position and size of the window are returned in twips.

Notes: The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

Windows

AppGetState (function)

Syntax AppGetState([*name\$*])

Description Returns an Integer specifying the state of the top-level window.

Comments The AppGetState function returns any of the following values:

If the window is	then AppGetState returns
Maximized	ebMaximized
Minimized	ebMinimized
Restored	ebRestored

The *name\$* parameter is a String containing the name of the desired application. If it is omitted, then the AppGetState function returns the name of the active application.

Examples

```
'This example saves the state of Program Manager, changes it, then
restores
'it to its original setting.

Sub Main()
    If AppFind$("Program Manager") = "" Then
        MsgBox "Can't find Program Manager."
        Exit Sub
    End If
    AppActivate "Program Manager"           'Activate Program Manager.
    state = AppGetState                      'Save its state.
    AppMinimize                             'Minimize it.
    MsgBox "Program Manager is now minimized. Select OK to restore it."
    AppActivate "Program Manager"
    AppSetState state                       'Restore it.
End Sub
```

See Also AppMaximize (statement); AppMinimize (statement); AppRestore (statement).

Platform(s) Windows.

Platform Notes: Under Windows, the *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppHide (statement)

Syntax AppHide [*name\$*]

Description Hides the named application.

Comments If the named application is already hidden, the `AppHide` statement will have no effect.

The *name\$* parameter is a `String` containing the name of the desired application. If it is omitted, then the `AppHide` statement hides the active application.

`AppHide` generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

Example 'This example hides Program Manager.

```
Sub Main()  
    'See whether Program Manager is running.  
    If AppFind$("Program Manager") = "" Then Exit Sub  
    AppHide "Program Manager"  
    MsgBox "Program Manager is now hidden. Press OK to show it once  
again."  
    AppShow "Program Manager"  
End Sub
```

See Also `AppShow` (statement).

Platform(s) Windows.

Platform Under Windows, the *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose

Notes: title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppList (statement)

Syntax `AppList AppNames$()`

Description Fills an array with the names of all open applications.

Comments The *AppNames\$* parameter must specify either a zero- or one-dimensional dynamic `String` array or a one-dimensional fixed `String` array. If the array is dynamic, then it will be redimensioned to match the number of open applications. For fixed arrays, `AppList` first erases each array element, then begins assigning application names to the elements in the array. If there are fewer elements than will fit in the array, then the remaining elements are unused. WM Basic returns a runtime error if the array is too small to hold the new elements.

After calling this function, you can use `LBound` and `UBound` to determine the new size of the array.

Example 'This example minimizes all applications on the desktop.

```
Sub Main()
    Dim apps$()
    AppList apps

    'Check to see whether any applications were found.
    If ArrayDims(apps) = 0 Then Exit Sub

    For i = LBound(apps) To UBound(apps)
        AppMinimize apps(i)
    Next i
End Sub
```

See Also WinList (statement).

Platform(s) Windows.

Platform Notes: Under Windows, the name of an application is considered to be the exact text that appears in the title bar of the application's main window.
Windows

AppMaximize (statement)

Syntax AppMaximize [*name\$*]

Description Maximizes the named application.

Comments The *name\$* parameter is a String containing the name of the desired application. If it is omitted, then the AppMaximize function maximizes the active application.

Example

```
Sub Main()
    AppMaximize "Program Manager"      'Maximize Program Manager.

    If AppFind$("NotePad") <> "" Then
        AppActivate "NotePad"          'Set the focus to NotePad.
        AppMaximize                    'Maximize it.
    End If
End Sub
```

See Also AppMinimize (statement); AppRestore (statement); AppMove (statement); AppSize (statement); AppClose (statement).

Platform(s) Windows.

Platform If the named application is maximized or hidden, the AppMaximize statement will have no effect.

Notes:

Windows The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppMaximize generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

AppMinimize (statement)

Syntax AppMinimize [*name\$*]

Description Minimizes the named application.

Comments The *name\$* parameter is a String containing the name of the desired application. If it is omitted, then the AppMinimize function minimizes the active application.

Example

```
Sub Main()  
    AppMinimize "Program Manager"      'Maximize Program Manager.  
  
    If AppFind$("NotePad") <> "" Then  
        AppActivate "NotePad"          'Set the focus to NotePad.  
        AppMinimize                     'Maximize it.  
    End If  
End Sub
```

See Also AppMaximize (statement); AppRestore (statement); AppMove (statement); AppSize (statement); AppClose (statement).

Platform(s) Windows.

Platform If the named application is minimized or hidden, the AppMinimize statement will have no effect.

Notes:

Windows The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppMinimize generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

AppMove (statement)

Syntax `AppMove X, Y [,name$]`

Description Sets the upper left corner of the named application to a given location.

Comments The AppMove statement takes the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the upper left corner of the new location of the application, relative to the upper left corner of the display.
<i>name\$</i>	String containing the name of the application to move. If this parameter is omitted, then the active application is moved.

Example 'This example activates Program Manager, then moves it 10 pixels to the 'right.

```
Sub Main()
    Dim x%,y%
    AppActivate "Program Manager"           'Activate Program Manager.
    AppGetPosition x%,y%,0,0                 'Retrieve its position.
    x% = x% + Screen.TwipsPerPixelX * 10    'Add 10 pixels.
    AppMove x% + 10,y%                       'Nudge it 10 pixels to the right.
End Sub
```

See Also AppMaximize (statement); AppMinimize (statement); AppRestore (statement); AppSize (statement); AppClose (statement).

Platform(s) Windows.

Platform Notes: If the named application is maximized or hidden, the AppMove statement will have no effect.

Windows The X and Y parameters are specified in twips.

AppMove will accept X and Y parameters that are off the screen.

The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppMove generates a runtime error if the named application is not enabled, as is the case if that application is currently displaying a modal dialog box.

AppRestore (statement)

Syntax `AppRestore [name$]`

Description Restores the named application.

Comments The *name\$* parameter is a String containing the name of the application to restore. If this parameter is omitted, then the active application is restored.

Example 'This example minimizes Program Manager, then restores it.

```
Sub Main()  
  If AppFind$("Program Manager") = "" Then Exit Sub  
  AppActivate "Program Manager"  
  AppMinimize "Program Manager"  
  MsgBox "Program Manager is now minimized. Press OK to restore it."  
  AppRestore "Program Manager"  
End Sub
```

See Also AppMaximize (statement); AppMinimize (statement); AppMove (statement); AppSize (statement); AppClose (statement).

Platform(s) Windows.

Platform Under Windows, the *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose

Notes: title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

Windows AppRestore will have an effect only if the main window of the named application is either maximized or minimized.

AppRestore will have no effect if the named window is hidden.

AppRestore will have no effect if the named window is hidden.

AppRestore generates a runtime error if the named application is not enabled, as is the case if that application is currently displaying a modal dialog box.

AppSetState (statement)

Syntax AppSetState *newstate* [, *name\$*]

Description Maximizes, minimizes, or restores the named application, depending on the value of *newstate*.

Comments The AppSetState statement takes the following parameters:

Parameter	Description								
<i>newstate</i>	Integer specifying the new state of the window. It can be any of the following values: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>ebMaximized</td><td>The named application is maximized.</td></tr> <tr> <td>ebMinimized</td><td>The named application is minimized.</td></tr> <tr> <td>ebRestored</td><td>The named application is restored.</td></tr> </table>	Value	Description	ebMaximized	The named application is maximized.	ebMinimized	The named application is minimized.	ebRestored	The named application is restored.
Value	Description								
ebMaximized	The named application is maximized.								
ebMinimized	The named application is minimized.								
ebRestored	The named application is restored.								
<i>name\$</i>	String containing the name of the application to change. If this parameter is omitted, then the active application is used.								

Example See AppGetState (function).

See Also AppGetState (function); AppMinimize (statement); AppMaximize (statement); AppRestore (statement).

Platform(s) Windows.

Platform Notes: Under Windows, the *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppShow (statement)

Syntax AppShow [*name\$*]

Description Makes the named application visible.

Comments The *name\$* parameter is a String containing the name of the application to show. If this parameter is omitted, then the active application is shown.

Example See AppHide (statement).

See Also AppHide (statement).

Platform(s) Windows.

Platform If the named application is already visible, AppShow will have no effect.

Notes:
Windows The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

AppShow generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

AppSize (statement)

Syntax AppSize *width,height* [,*name\$*]

Description Sets the width and height of the named application.

Comments The AppSize statement takes the following parameters:

Parameter	Description
<i>width,height</i>	Integer coordinates specifying the new size of the application.
<i>name\$</i>	String containing the name of the application to resize. If this parameter is omitted, then the active application is used.

Example 'This example enlarges the active application by 10 pixels in both the 'vertical and horizontal directions.

```
Sub Main()  
    Dim w%,h%  
    AppGetPosition 0,0,w%,h%           'Get current width/height.  
    x% = x% + Screen.TwipsPerPixelX * 10 'Add 10 pixels.  
    y% = y% + Screen.TwipsPerPixelY * 10 'Add 10 pixels.  
    AppSize w%,h%                     'Change to new size.  
End Sub
```

See Also AppMaximize (statement); AppMinimize (statement); AppRestore (statement); AppMove (statement); AppClose (statement).

Platform(s) Windows.

Platform The *width* and *height* parameters are specified in twips.
Notes: This statement will only work if the named application is restored (i.e., not
Windows minimized or maximized).

The *name\$* parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

A runtime error results if the application being resized is not enabled, which is the case if that application is displaying a modal dialog box when an `AppSize` statement is executed.

AppType (function)

Syntax `AppType [(name$)]`

Description Returns an `Integer` indicating the executable file type of the named application:

`ebDos` DOS executable

`ebWindows` Windows executable

Comments The *name\$* parameter is a `String` containing the name of the application. If this parameter is omitted, then the active application is used.

Example 'This example creates an array of strings containing the names of all the running Windows applications. It uses the AppType command to determine whether an application is a Windows application or a DOS application.

```
Sub Main()  
    Dim apps$(),wapps$()  
  
    AppList apps      'Retrieve a list of all Windows and DOS apps.  
    If ArrayDims(apps) = 0 Then  
        MsgBox "There are no running applications."  
        Exit Sub  
    End If  
  
    'Create an array to hold only the Windows apps.  
    ReDim wapps$(UBound(apps))  
    n = 0 'Copy the Windows apps from one array to the target array.  
    For i = LBound(apps) to UBound(apps)  
        If AppType(apps(i)) = ebWindows Then  
            wapps(n) = apps(i)  
            n = n + 1  
        End If  
    Next i  
  
    If n = 0 Then 'Make sure at least one Windows app was found.  
        MsgBox "There are no running Windows applications."  
        Exit Sub  
    End If  
  
    ReDim Preserve wapps(n - 1) 'Resize to hold the exact number.  
    'Let the user pick one.  
    index% = SelectBox("Windows Applications","Select a Windows  
application:",wapps)  
End Sub
```

See Also AppFilename\$ (function).

Platform(s) Windows.

Platform Under Windows, the *name\$* parameter is the exact string appearing in the title

Notes: bar of the named application's main window. If no application is found whose

Windows title exactly matches *name\$*, then a second search is performed for applications whose title string begins with *name\$*. If more than one application is found that matches *name\$*, then the first application encountered is used.

ArrayDims (function)

Syntax ArrayDims (*arrayvariable*)

Description Returns an Integer containing the number of dimensions of a given array.

Comments This function can be used to determine whether a given array contains any elements or if the array is initially created with no dimensions and then redimensioned by another function, such as the `FileList` function, as shown in the following example.

Example 'This example allocates an empty (null-dimensioned) array; fills the array
'with a list of filenames, which resizes the array; then tests the array
'dimension and displays an appropriate message.

```
Sub Main()  
    Dim f$()  
    FileList f$, "c:\*.bat"  
    If ArrayDims(f$) = 0 Then  
        MsgBox "The array is empty."  
    Else  
        MsgBox "The array size is: " & (UBound(f$) - UBound(f$) + 1)  
    End If  
End Sub
```

See Also `LBound` (function); `UBound` (function); Arrays (topic).

Platform(s) Windows and Macintosh.

Arrays (topic)

Declaring Array Variables

Arrays in WM Basic are declared using any of the following statements:

```
Dim
Public
Private
```

For example:

```
Dim a(10) As Integer
Public LastNames(1 to 5,-2 to 7) As Variant
Private
```

Arrays of any data type can be created, including Integer, Long, Single, Double, Boolean, Date, Variant, Object, user-defined structures, and data objects.

The lower and upper bounds of each array dimension must be within the following range:

$-32768 \leq bound \leq 32767$

Arrays can have up to 60 dimensions.

Arrays can be declared as either fixed or dynamic, as described below.

Fixed Arrays

The dimensions of fixed arrays cannot be adjusted at execution time. Once declared, a fixed array will always require the same amount of storage. Fixed arrays can be declared with the `Dim`, `Private`, or `Public` statement by supplying explicit dimensions. The following example declares a fixed array of ten strings:

```
Dim a(10) As String
```

Fixed arrays can be used as members of user-defined data types. The following example shows a structure containing fixed-length arrays:

```
Type Foo
    rect(4) As Integer
    colors(10) As Integer
End Type
```

Only fixed arrays can appear within structures.

Dynamic Arrays

Dynamic arrays are declared without explicit dimensions, as shown below:

```
Public Ages() As Integer
```

Dynamic arrays can be resized at execution time using the `Redim` statement:

```
Redim Ages$(100)
```

Subsequent to their initial declaration, dynamic arrays can be redimensioned any number of times. When redimensioning an array, the old array is first erased unless you use the `Preserve` keyword, as shown below:

```
Redim Preserve Ages$(100)
```

Dynamic arrays cannot be members of user-defined data types.

Passing Arrays

Arrays are always passed by reference.

Querying Arrays

The following table describes the functions used to retrieve information about arrays.

Use this function to

<code>LBound</code>	Retrieve the lower bound of an array. A runtime error is generated if the array has no dimensions.
<code>UBound</code>	Retrieve the upper bound of an array. A runtime error is generated if the array has no dimensions.
<code>ArrayDims</code>	Retrieve the number of dimensions of an array. This function returns 0 if the array has no dimensions.

Operations on Arrays

The following table describes the function that operate on arrays:

Use this command	to
<code>ArraySort</code>	Sort an array of integers, longs, singles, doubles, currency, Booleans, dates, or variants.
<code>FileList</code>	Fill an array with a list of files in a given directory.
<code>DiskDrives</code>	Fill an array with a list of valid drive letters.
<code>AppList</code>	Fill an array with a list of running applications.
<code>WinList</code>	Fill an array with a list of top-level windows.
<code>SelectBox</code>	Display the contents of an array in a list box.

76 Working Model Basic User's Manual

PopupMenu	Display the contents of an array in a pop-up menu.
ReadIniSection	Fill an array with the item names from a section in an ini file.
FileDirs	Fill an array with a list of subdirectories.
Erase	Erase all the elements of an array.
ReDim	Establish the bounds and dimensions of an array.
Dim	Declare an array.

ArraySort (statement)

Syntax `ArraySort array()`

Description Sorts a single-dimensioned array in ascending order.

Comments If a string array is specified, then the routine sorts alphabetically in ascending order using case-sensitive string comparisons. If a numeric array is specified, the `ArraySort` statement sorts smaller numbers to the lowest array index locations.

WM Basic generates a runtime error if you specify an array with more than one dimension.

When sorting an array of variants, the following rules apply:

- A runtime error is generated if any element of the array is an object.
- `String` is greater than any numeric type.
- `Null` is less than `String` and all numeric types.
- `Empty` is treated as a number with the value 0.
- String comparison is case-sensitive (this function is not affected by the `Option Compare` setting).

Example 'This example dimensions an array and fills it with filenames using `FileList`,
'then sorts the array and displays it in a select box.

```
Sub Main()  
    Dim f$()  
    FileList f$,"c:\*.*"   
    ArraySort f$  
    r% = SelectBox("Files","Choose one:",f$)  
End Sub
```

See Also `ArrayDims` (function); `LBound` (function); `UBound` (function).

Platform(s) Windows and Macintosh.

Asc (function)

Syntax `Asc(text)`

Description Returns an Integer containing the numeric code for the first character of *text*.

Comments The return value is an integer between 0 and 255.

Example 'This example fills an array with the ASCII values of the string *s* components
'and displays the result.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    s$ = InputBox("Please enter a string.", "Enter String")
    If s$ = "" Then End      'Exit if no string entered.
    For i = 1 To Len(s$)
        msg = msg & Asc(Mid$(s$, i, 1)) & crlf
    Next i
    MsgBox "The Asc values of the string are:" & msg
End Sub
```

See Also `Chr`, `Chr$` (functions).

Platform(s) Windows and Macintosh.

AskBox\$ (function)

Syntax `AskBox$(prompt [, default])`

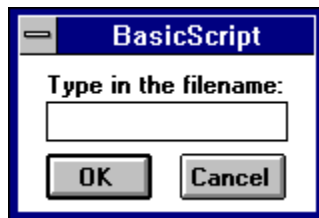
Description Displays a dialog box requesting input from the user and returns that input as a String.

78 Working Model Basic User's Manual

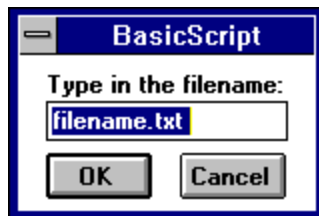
Comments The AskBox\$ function takes the following parameters:

Parameter	Description
<i>prompt\$</i>	String containing the text to be displayed above the text box. The dialog box is sized to the appropriate width depending on the width of <i>prompt\$</i> . A runtime error is generated if <i>prompt\$</i> is Null.
<i>default\$</i>	String containing the initial content of the text box. The user can return the default by immediately selecting OK. A runtime error is generated if <i>default\$</i> is Null. The AskBox\$ function returns a String containing the input typed by the user in the text box. A zero-length string is returned if the user selects Cancel. When the dialog box is displayed, the text box has the focus. The user can type a maximum of 255 characters into the text box displayed by AskBox\$.

```
s$ = AskBox$("Type in the filename: ")
```



```
s$ = AskBox$("Type in the filename:", "filename.txt")
```



Example 'This example asks the user to enter a filename and then displays what he or she has typed.

```
Sub Main()  
    s$ = AskBox$("Type in the filename:")  
    MsgBox "The filename was: " & s$  
End Sub
```

See Also MsgBox (statement); AskPassword\$ (function); InputBox, InputBox\$ (functions); OpenFilename\$ (function); SaveFilename\$ (function); SelectBox (function).

Platform(s) Windows and Macintosh.

Platform The text in the dialog box is displayed in 8-point MS Sans Serif.

Notes:
Windows

AskPassword\$ (function)

Syntax AskPassword\$(*prompt\$*)

Description Returns a String containing the text that the user typed.

Comments Unlike the AskBox\$ function, the user sees asterisks in place of the characters that are actually typed. This allows the hidden input of passwords.

The *prompt\$* parameter is a String containing the text to appear above the text box. The dialog box is sized to the appropriate width depending on the width of *prompt\$*.

When the dialog box is displayed, the text box has the focus.

A maximum of 255 characters can be typed into the text box.

A zero-length string is returned if the user selects Cancel.

```
s$ = AskPassword$("Type in the password:")
```



Example

```
Sub Main()  
    s$ = AskPassword$("Type in the password:")  
    MsgBox "The password entered is: " & s$  
End Sub
```

See Also MsgBox (statement); AskBox\$ (function); InputBox, InputBox\$ (functions); OpenFilename\$ (function); SaveFilename\$ (function); SelectBox (function); AnswerBox (function).

Platform(s) Windows and Macintosh.

Platform The text in the dialog box is displayed in 8-point MS Sans Serif.

Notes:
Windows

Atn (function)

Syntax `Atn(number)`

Description Returns the angle (in radians) whose tangent is *number*.

Comments Some helpful conversions:

- Pi (3.1415926536) radians = 180 degrees.
- 1 radian = 57.2957795131 degrees.
- 1 degree = .0174532925 radians.

Example 'This example finds the angle whose tangent is 1 (45 degrees) and displays
'the result.

```
Sub Main()  
    a# = Atn(1.00)  
    MsgBox "1.00 is the tangent of " & a# & " radians (45 degrees)."  
End Sub
```

See Also Tan (function); Sin (function); Cos (function).

Platform(s) Windows and Macintosh.

Basic.Capability (method)

Syntax `Basic.Capability(which)`

Description Returns True if the specified capability exists on the current platform; returns False otherwise.

Comments The *which* parameter is an Integer specifying the capability for which to test. It can be any of the following values:

Value	Returns True If the Platform Supports
1	Disk drives
2	System file attribute (ebSystem)
3	Hidden file attribute (ebHidden)
4	Volume label file attribute (ebVolume)
5	Archive file attribute (ebArchive)
6	Denormalized floating-point math
7	File locking (i.e., the Lock and Unlock statements)
8	Big endian byte ordering

Example 'This example tests to see whether your current platform supports disk drives and hidden file attributes and displays the result.

```
Sub Main()
    msg = "This operating system "

    If Basic.Capability(1) Then
        msg = msg & "supports disk drives."
    Else
        msg = msg & "does not support disk drives."
    End If

    MsgBox msg
End Sub
```

See Also Cross-Platform Scripting (topic); **Basic.OS** (property).

Platform(s) Windows and Macintosh.

Basic.Eoln\$ (property)

Syntax **Basic.Eoln\$**

Description Returns a **String** containing the end-of-line character sequence appropriate to the current platform.

Comments This string will be either a carriage return, a carriage return/line feed, or a line feed.

Example 'This example writes two lines of text in a message box.

```
Sub Main()
    MsgBox "This is the first line of text." & Basic.Eoln$ & "This is the second line of text."
End Sub
```

See Also Cross-Platform Scripting (topic); **Basic.PathSeparator\$** (property).

Platform(s) Windows and Macintosh.

Basic.FreeMemory (property)

Syntax **Basic.FreeMemory**

Description Returns a **Long** representing the number of bytes of free memory in WM Basic's data space.

Comments This function returns the size of the largest free block in WM Basic's data space. Before this number is returned, the data space is compacted, consolidating free space into a single contiguous free block.

WM Basic's data space contains strings and dynamic arrays.

82 Working Model Basic User's Manual

Example 'This example displays free memory in a dialog box.

```
Sub Main()  
    MsgBox "The largest free memory block is: " & Basic.FreeMemory  
End Sub
```

See Also `System.TotalMemory (property)`; `System.FreeMemory (property)`;
`System.FreeResources (property)`; `Basic.FreeMemory (property)`.

Platform(s) Windows and Macintosh.

Basic.HomeDir\$ (property)

Syntax `Basic.HomeDir$`

Description Returns a `String` specifying the directory containing WM Basic.

Comments This method is used to find the directory in which the WM Basic files are located.

Example 'This example assigns the home directory to HD and displays it.

```
Sub Main()  
    hd$ = Basic.HomeDir$  
    MsgBox "The WM Basic home directory is: " & hd$  
End Sub
```

See Also `System.WindowsDirectory$ (property)`.

Platform(s) Windows and Macintosh.

Basic.OS (property)

Syntax `Basic.OS`

Description Returns an `Integer` indicating the current platform.

Comments

Value	Constant	Platform
-------	----------	----------

0	<code>ebWin16</code>	Microsoft Windows
---	----------------------	-------------------

10	<code>ebMacintosh</code>	Apple Macintosh
----	--------------------------	-----------------

The value returned is not necessarily the platform under which WM Basic is running but rather an indicator of the platform for which WM Basic was created. For example, it is possible to run WM Basic for Windows under Windows NT Workstation. In this case, `Basic.OS` will return 0.

Example 'This example determines the operating system for which this version was created and displays the appropriate message.

```
Sub Main()
    Select Case Basic.OS
        Case ebWin16
            s = "Windows"
        Case ebMacintosh
            s = "Macintosh"
        Case Else
            s = "neither Windows nor Macintosh"
    End Select
    MsgBox "You are currently running " & s
End Sub
```

See Also Cross-Platform Scripting (topic).

Platform(s) Windows and Macintosh.

Basic.PathSeparator\$ (property)

Syntax Basic.PathSeparator\$

Description Returns a String containing the path separator appropriate for the current platform.

Comments The returned string is any one of the following characters: / (slash), \ (back slash), : (colon)

Example

```
Sub Main()
    MsgBox "The path separator for this platform is: " &
        Basic.PathSeparator$
End Sub
```

See Also Basic.Eoln\$ (property); Cross-Platform Scripting (topic).

Platform(s) Windows and Macintosh.

Basic.Version\$ (property)

Syntax Basic.Version\$

Description Returns a String containing the version of WM Basic.

Comments This function returns the major and minor version numbers in the format *major.minor.BuildNumber*, as in "2.00.30."

Example 'This example displays the current version of WM Basic.

```
Sub Main()
    MsgBox "Version " & Basic.Version$ & " of WM Basic is running"
End Sub
```

Platform(s) Windows and Macintosh.

Beep (statement)

Syntax Beep

Description Makes a single system beep.

Example 'This example causes the system to beep five times and displays a reminder message.

```
Sub Main()  
    For i = 1 To 5  
        Beep  
        Sleep(200)  
    Next i  
    MsgBox "You have an upcoming appointment!"  
End Sub
```

See Also Mci (function).

Platform(s) Windows and Macintosh.

Begin Dialog (statement)

Syntax Begin Dialog *DialogName* [*x*],[*y*],[*width*],[*height*],[*title\$*] [, [*DlgProc*] [, [*PicName\$*] [, [*style*]]]
 Dialog Statements
End Dialog

Description Defines a dialog box template for use with the Dialog statement and function.

Comments A dialog box template is constructed by placing any of the following statements between the Begin Dialog and End Dialog statements (no other statements besides comments can appear within a dialog box template):

Picture	OptionButton	OptionGroup
CancelButton		Text TextBox
GroupBox	DropListBox	ListBox
ComboBox	CheckBox	PictureButton
PushButton	OKButton	

The Begin Dialog statement requires the following parameters:

Parameter	Description
<i>x, y</i>	Integer coordinates specifying the position of the upper left corner of the dialog box relative to the parent window. These coordinates are in dialog units. If either coordinate is unspecified, then the dialog box will be centered in that direction on the parent window.
<i>width, height</i>	Integer coordinates specifying the width and height of the dialog box (in dialog units).
<i>DialogName</i>	Name of the dialog box template. Once a dialog box template has been created, a variable can be dimensioned using this name.
<i>title\$</i>	String containing the name to appear in the title bar of the dialog box. If this parameter specifies a zero-length string, then the name "WM Basic" is used.
<i>.DlgProc</i>	Name of the dialog function. The routine specified by <i>.DlgProc</i> will be called by WM Basic when certain actions occur during processing of the dialog box. (See <code>DlgProc [prototype]</code> for additional information about dialog functions.) If this omitted, then WM Basic processes the dialog box using the default dialog box processing behavior.
<i>PicName\$</i>	String specifying the name of a DLL containing pictures. This DLL is used as the origin for pictures when the picture type is 10. If omitted, then no picture library will be used.
<i>style</i>	Specifies extra styles for the dialog. It can be any of the following values:

Value	Meaning
0	Dialog does not contain a title or close box.
1	Dialog contains a title and no close box.
2 (or omitted)	Dialog contains both the title and close box.

WM Basic generates an error if the dialog box template contains no controls.

A dialog box template must have at least one `PushButton`, `OKButton`, or `CancelButton` statement. Otherwise, there will be no way to close the dialog box.

Dialog units are defined as 1/4 the width of the font in the horizontal direction and 1/8 the height of the font in the vertical direction.

Any number of user dialog boxes can be created, but each one must be created using a different name as the *DialogName*. Only one user dialog box may be invoked at any time.

Expression Evaluation within the Dialog Box Template

The `Begin Dialog` statement creates the template for the dialog box. Any expression or variable name that appears within any of the statements in the dialog box template is not evaluated until a variable is dimensioned of type *DialogName*. The following example shows this behavior:

```
MyTitle$ = "Hello, World"
Begin Dialog MyTemplate 16,32,116,64,MyTitle$
    OKButton 12,40,40,14
End Dialog
MyTitle$ = "Sample Dialog"
Dim Dummy As MyTemplate
rc% = Dialog(Dummy)
```

The above example creates a dialog box with the title "Sample Dialog".

Expressions within dialog box templates cannot reference external subroutines or functions.

All controls within a dialog box use the same font. The fonts used for text and text box control can be changed explicitly by setting the font parameters in the `Text` and `TextBox` statements. A maximum of 128 fonts can be used within a single dialog, although the practical limitation may be less.

Example 'This example creates an exit dialog box.

```
Sub Main()
    Begin Dialog QuitDialogTemplate 16,32,116,64,"Quit"
        Text 4,8,108,8,"Are you sure you want to exit?"
        CheckBox 32,24,63,8,"Save Changes",.SaveChanges
        OKButton 12,40,40,14
        CancelButton 60,40,40,14
    End Dialog
    Dim QuitDialog As QuitDialogTemplate
    rc% = Dialog(QuitDialog)
End Sub
```

See Also `CancelButton (statement)`; `CheckBox (statement)`; `ComboBox (statement)`; `Dialog (function)`; `Dialog (statement)`; `DropListBox (statement)`; `GroupBox (statement)`; `ListBox (statement)`; `OKButton (statement)`; `OptionButton (statement)`; `OptionGroup (statement)`; `Picture (statement)`; `PushButton (statement)`; `Text (statement)`; `TextBox (statement)`; `DlgProc (function)`.

Platform(s) Windows and Macintosh.

Within a dialog box, the default font is 8-point MS Sans Serif.

Platform Notes:
Windows

Platform Within user dialog boxes, the default font is 10-point Geneva.
Notes:
Macintosh

Boolean (data type)

Syntax `Boolean`

Description A data type capable of representing the logical values `True` and `False`.

Comments Boolean variables are used to hold a binary value—either `True` or `False`. Variables can be declared as `Boolean` using the `Dim`, `Public`, or `Private` statement.

Variants can hold `Boolean` values when assigned the results of comparisons or the constants `True` or `False`.

Internally, a `Boolean` variable is a 2-byte value holding `-1` (for `True`) or `0` (for `False`).

Any type of data can be assigned to `Boolean` variables. When assigning, non-0 values are converted to `True`, and 0 values are converted to `False`.

When appearing as a structure member, `Boolean` members require 2 bytes of storage.

When used within binary or random files, 2 bytes of storage are required.

When passed to external routines, `Boolean` values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack.

There is no type-declaration character for `Boolean` variables.

`Boolean` variables that have not yet been assigned are given an initial value of `False`.

See Also `Currency` (data type); `Date` (data type); `Double` (data type); `Integer` (data type); `Long` (data type); `Object` (data type); `Single` (data type); `String` (data type); `Variant` (data type); `DefType` (statement); `CBool` (function); `True` (constant); `False` (constant).

Platform(s) Windows and Macintosh.

ButtonEnabled (function)

Syntax `ButtonEnabled(name$ | id)`

88 Working Model Basic User's Manual

Description Returns `True` if the specified button within the current window is enabled; returns `False` otherwise.

Comments The `ButtonEnabled` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the push button.
<i>id</i>	Integer specifying the ID of the push button. When a button is enabled, it can be clicked using the <code>SelectButton</code> statement.
Note: The <code>ButtonEnabled</code> function is used to determine whether a push button is enabled in another application's dialog box. Use the <code>DlgEnable</code> function to retrieve the enabled state of a push button in a dynamic dialog box.	

Example 'This code fragment checks to see whether a button is enabled before clicking it.

```
Sub Main()  
    If ButtonEnabled("Browse...") Then  
        SelectButton "Browse..."  
    Else  
        MsgBox "Can't browse right now."  
    End If  
End Sub
```

See Also `ButtonExists` (function); `SelectButton` (statement).

Platform(s) Windows.

ButtonExists (function)

Syntax `ButtonExists(name$ | id)`

Description Returns `True` if the specified button exists within the current window; returns `False` otherwise.

Comments The `ButtonExists` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the push button.
<i>id</i>	Integer specifying the ID of the push button.
Note: The <code>ButtonExists</code> function is used to determine whether a push button exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example 'This code fragment selects the More button if it exists. If it does not exist, then this code fragment does nothing.

```
Sub Main()
    If ButtonExists("More >>") Then
        SelectButton "More >>"      'Display more stuff.
    End If
End Sub
```

See Also ButtonEnabled (function); SelectButton (statement).

Platform(s) Windows.

ByRef (keyword)

Syntax ... ,ByRef *parameter* , ...

Description Used within the Sub...End Sub, Function...End Function, Or Declare statement to specify that a given parameter can be modified by the called routine.

Comments Passing a parameter by reference means that the caller can modify that variable's value.

Unlike the ByVal keyword, the ByRef keyword cannot be used when passing a parameter. The absence of the ByVal keyword is sufficient to force a parameter to be passed by reference:

```
MySub ByVal i      'Pass i by value.
MySub ByRef i      'Illegal (will not compile).
MySub i            'Pass i by reference.
```

Example

```
Sub Test(ByRef a As Variant)
    a = 14
End Sub

Sub Main()
    b = 12
    Test b
    MsgBox "The ByRef value is: " & b      'Displays 14.
End Sub
```

See Also () (keyword), ByVal (keyword).

Platform(s) Windows and Macintosh.

ByVal (keyword)

Syntax ...ByVal *parameter*...

Description Forces a parameter to be passed by value rather than by reference.

Comments The `ByVal` keyword can appear before any parameter passed to any function, statement, or method to force that parameter to be passed by value. Passing a parameter by value means that the caller cannot modify that variable's value.

Enclosing a variable within parentheses has the same effect as the `ByVal` keyword:

```
Foo ByVal i           'Forces i to be passed by value.  
Foo(i)               'Forces i to be passed by value.
```

When calling external statements and functions (i.e., routines defined using the `Declare` statement), the `ByVal` keyword forces the parameter to be passed by value regardless of the declaration of that parameter in the `Declare` statement. The following example shows the effect of the `ByVal` keyword used to pass an `Integer` to an external routine:

```
Declare Sub Foo Lib "MyLib" (ByRef i As Integer)  
  
i% = 6  
Foo ByVal i%           'Pass a 2-byte Integer.  
Foo i%                 'Pass a 4-byte pointer to an Integer.
```

Since the `Foo` routine expects to receive a pointer to an `Integer`, the first call to `Foo` will have unpredictable results.

Example 'This example demonstrates the use of the `ByVal` keyword.

```
Sub Foo(a As Integer)  
    a = a + 1  
End Sub  
  
Sub Main()  
    Dim i As Integer  
    i = 10  
    Foo i  
    MsgBox "The ByVal value is: " & i    'Displays 11 (Foo changed the  
value).  
    Foo ByVal i  
    MsgBox "The Byval value is still: " & i    'Displays 11 (Foo did  
not change the value).  
End Sub
```

See Also `()` (keyword), `ByRef` (keyword).

Platform(s) Windows and Macintosh.

Call (statement)

Syntax `Call subroutine_name [(arguments)]`

Description Transfers control to the given subroutine, optionally passing the specified arguments.

Comments Using this statement is equivalent to:

subroutine_name [*arguments*]

Use of the Call statement is optional. The call statement can only be used to execute subroutines; functions cannot be executed with this statement. The subroutine to which control is transferred by the Call statement must be declared outside of the Main procedure, as shown in the following example.

Example

```
'This example demonstrates the use of the Call statement to pass
control to
'another function.

Sub Example_Call(s$)
    'This subroutine is declared externally to Main and displays the
    text
    'passed in the parameter s$.
    MsgBox "Call: " & s$
End Sub

Sub Main()
    'This example assigns a string variable to display, then calls
    subroutine
    'Example_Call, passing parameter S$ to be displayed in a message
    box
    'within the subroutine.
    s$ = "DAVE"
    Example_Call s$
    Call Example_Call("SUSAN")
End Sub
```

See Also Goto (statement); GoSub (statement); Declare (statement).

Platform(s) Windows and Macintosh.

CancelButton (statement)

Syntax CancelButton *X, Y, width, height* [, *.Identifier*]

Description Defines a Cancel button that appears within a dialog box template.

Comments This statement can only appear within a dialog box template (i.e., between the `Begin Dialog` and `End Dialog` statements).

Selecting the Cancel button (or pressing Esc) dismisses the user dialog box, causing the `Dialog` function to return 0. (Note: A dialog function can redefine this behavior.) Pressing the Esc key or double-clicking the close box will have no effect if a dialog box does not contain a `CancelButton` statement.

The `CancelButton` statement requires the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>.Identifier</i>	Optional parameter specifying the name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>). If omitted, then the word <code>Cancel</code> is used.
	A dialog box must contain at least one <code>OKButton</code> , <code>CancelButton</code> , or <code>PushButton</code> statement; otherwise, the dialog box cannot be dismissed.

Example 'This example creates a dialog box with OK and Cancel buttons.

```
Sub Main()  
  Begin Dialog SampleDialogTemplate 37,32,48,52,"Sample"  
    OKButton 4,12,40,14,.OK  
    CancelButton 4,32,40,14,.Cancel  
  End Dialog  
  Dim SampleDialog As SampleDialogTemplate  
  r% = Dialog(SampleDialog)  
  If r% = 0 Then MsgBox "Cancel was pressed!"  
End Sub
```

See Also `CheckBox (statement)`; `ComboBox (statement)`; `Dialog (function)`; `Dialog (statement)`; `DropListBox (statement)`; `GroupBox (statement)`; `ListBox (statement)`; `OKButton (statement)`; `OptionButton (statement)`; `OptionGroup (statement)`; `Picture (statement)`; `PushButton (statement)`; `Text (statement)`; `TextBox (statement)`; `Begin Dialog (statement)`, `PictureButton (statement)`.

Platform(s) Windows and Macintosh.

CBool (function)

Syntax `CBool (expression)`

Description Converts *expression* to True or False, returning a Boolean value.

Comments The *expression* parameter is any expression that can be converted to a Boolean. A runtime error is generated if *expression* is Null.

All numeric data types are convertible to Boolean. If *expression* is zero, then the CBool returns False; otherwise, CBool returns True. Empty is treated as False.

If *expression* is a String, then CBool first attempts to convert it to a number, then converts the number to a Boolean. A runtime error is generated if *expression* cannot be converted to a number.

A runtime error is generated if *expression* cannot be converted to a Boolean.

Example 'This example uses CBool to determine whether a string is numeric
'or just plain text.

```
Sub Main()
    Dim IsNumericOrDate As Boolean
    s$ = "34224.54"
    IsNumericOrDate = CBool(IsNumeric(s$) Or IsDate(s$))
    If IsNumericOrDate = True Then
        MsgBox s$ & " is either a valid date or number!"
    Else
        MsgBox s$ & " is not a valid date or number!"
    End If
End Sub
```

See Also CCur (function); CDate, CVDate (functions); CDBl (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVerErr (function); Boolean (data type).

Platform(s) Windows and Macintosh.

CCur (function)

Syntax CCur (*expression*)

Description Converts any expression to a Currency.

Comments This function accepts any expression convertible to a Currency, including strings. A runtime error is generated if *expression* is Null or a String not convertible to a number. Empty is treated as 0.

When passed a numeric expression, this function has the same effect as assigning the numeric expression *number* to a Currency.

When used with variants, this function guarantees that the variant will be assigned a Currency (VarType 6).

Example 'This example displays the value of a String converted into a Currency value.

```
Sub Main()  
    i$ = "100.44"  
    MsgBox "The currency value is: " & CCur(i$)  
End Sub
```

See Also CBool (function); CDate, CVDate (functions); CDb1 (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVer (function); Currency (data type).

Platform(s) Windows and Macintosh.

CDate, CVDate (functions)

Syntax CDate(*expression*)
CVDate(*expression*)

Description Converts *expression* to a date, returning a Date value.

Comments The *expression* parameter is any expression that can be converted to a Date. A runtime error is generated if *expression* is Null.

If *expression* is a String, an attempt is made to convert it to a Date using the current country settings. If *expression* does not represent a valid date, then an attempt is made to convert *expression* to a number. A runtime error is generated if *expression* cannot be represented as a date.

These functions are sensitive to the date and time formats of your computer.

The CDate and CVDate functions are identical.

Example 'This example takes two dates and computes the difference between them.

```
Sub Main()  
    Dim date1 As Date  
    Dim date2 As Date  
    Dim diff As Date  
  
    date1 = CDate("#1/1/1994#")  
    date2 = CDate("February 1, 1994")  
    diff = DateDiff("d", date1, date2)  
  
    MsgBox "The date difference is " & CInt(diff) & " days."  
End Sub
```

See Also CCur (function); CBool (function); CDb1 (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVer (function); Date (data type).

Platform(s) Windows and Macintosh.

CDbl (function)

Syntax CDb1 (*expression*)

Description Converts any expression to a Double.

Comments This function accepts any expression convertible to a Double, including strings. A runtime error is generated if *expression* is Null. Empty is treated as 0.0.

When passed a numeric expression, this function has the same effect as assigning the numeric expression *number* to a Double.

When used with variants, this function guarantees that the variant will be assigned a Double (VarType 5).

Example 'This example displays the result of two numbers as a Double.

```
Sub Main()
    i% = 100
    j! = 123.44
    MsgBox "The double value is: " & CDb1(i% * j!)
End Sub
```

See Also CCur (function); CBool (function); CDate, CVDate (functions); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); CVer (function); Double (data type).

Platform(s) Windows and Macintosh.

ChDir (statement)

Syntax ChDir *newdir\$*

Description Changes the current directory of the specified drive to *newdir\$*.

This routine will not change the current drive. (See ChDrive [statement].)

Example 'This example saves the current directory, then changes to the root directory, displays the old and new directories, restores the old directory, and displays it.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    save$ = CurDir$
    ChDir (Basic.PathSeparator)
    MsgBox "Old: " & save$ & crlf & "New: " & CurDir$
    ChDir (save$)
    MsgBox "Directory restored to: " & CurDir$
End Sub
```

See Also ChDrive (statement); CurDir, CurDir\$ (functions); Dir, Dir\$ (functions); MkDir (statement); Rmdir (statement); DirList (statement).

Platform(s) Windows and Macintosh.

Platform The Macintosh does not support drive letters.

Notes: The Macintosh uses the colon (":") as the path separator. A double colon ("::") specifies the parent directory.
Macintosh

ChDrive (statement)

Syntax `ChDrive DriveLetter$`

Description Changes the default drive to the specified drive.

Comments Only the first character of *DriveLetter\$* is used.

DriveLetter\$ is not case-sensitive.

 If *DriveLetter\$* is empty, then the current drive is not changed.

Example 'This example saves the current directory in CD, then extracts the current
 'drive letter and saves it in Save\$. If the current drive is D, then it
 is
 'changed to C; otherwise, it is changed to D. Then the saved drive is
 'restored and displayed.

```
Const crlf$ = Chr$(13) + Chr$(10)
```

```
Sub Main()  
  cd$ = CurDir$  
  save$ = Mid$(CurDir$,1,1)  
  If save$ = "D" Then  
    ChDrive("C")  
  Else  
    ChDrive("D")  
  End If  
  MsgBox "Old: " & save$ & crlf & "New: " & CurDir$  
  ChDrive (save$)  
  MsgBox "Directory restored to: " & CurDir$  
End Sub
```

See Also `ChDir (statement)`; `CurDir`, `CurDir$ (functions)`; `Dir`, `Dir$ (functions)`;
 `MkDir (statement)`; `Rmdir (statement)`; `DiskDrives (statement)`.

Platform(s) Windows.

Platform Macintosh does not support drive letters.

Notes:
Macintosh

CheckBox (statement)

Syntax `CheckBox X, Y, width, height, title$, .Identifier`

Description Defines a check box within a dialog box template.

Comments Check box controls are either on or off, depending on the value of *.Identifier*.
This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).

The CheckBox statement requires the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>title\$</i>	String containing the text that appears within the check box. This text may contain an ampersand character to denote an accelerator letter, such as "&Font " for Font (indicating that the Font control may be selected by pressing the F accelerator key).
<i>.Identifier</i>	Name by which this control can be referenced by statements in a dialog function (such asDlgFocus andDlgEnable). This parameter also creates an integer variable whose value corresponds to the state of the check box (1 = checked; 0 = unchecked). This variable can be accessed using the syntax:

DialogVariable.Identifier.

When the dialog box is first created, the value referenced by *.Identifier* is used to set the initial state of the check box. When the dialog box is dismissed, the final state of the check box is placed into this variable. By default, the *.Identifier* variable contains 0, meaning that the check box is unchecked.

Example 'This example displays a dialog box with two check boxes in different states.

```
Sub Main()
    Begin Dialog SaveOptionsTemplate 36,32,151,52,"Save"
        GroupBox 4,4,84,40,"GroupBox"
        CheckBox 12,16,67,8,"Include heading",.IncludeHeading
        CheckBox 12,28,73,8,"Expand keywords",.ExpandKeywords
        OKButton 104,8,40,14,.OK
        CancelButton 104,28,40,14,.Cancel
    End Dialog
    Dim SaveOptions As SaveOptionsTemplate
    SaveOptions.IncludeHeading = 1      'Check box initially on.
    SaveOptions.ExpandKeywords = 0     'Check box initially off.
    r% = Dialog(SaveOptions)
    If r% = -1 Then
        MsgBox "OK was pressed."
    End If
End Sub
```

See Also CancelButton (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Platform(s) Windows and Macintosh.

Platform Notes: On Windows, accelerators are underlined, and the accelerator combination Alt+*letter* is used.

Windows

Platform Notes: On the Macintosh, accelerators are normal in appearance, and the accelerator combination Command+*letter* is used.

Macintosh

CheckBoxEnabled (function)

Syntax CheckBoxEnabled(*name\$* | *id*)

Description Returns True if the specified check box within the current window is enabled; returns False otherwise.

Comments The CheckBoxEnabled function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the check box.
<i>id</i>	Integer specifying the ID of the check box. When a check box is enabled, its state can be set using the SetCheckBox statement.
Note: The CheckBoxEnabled function is used to determine whether a check box is enabled in another application's dialog box. Use the DlgEnable function within dynamic dialog boxes.	

Example 'This code checks to see whether a check box is enabled.

```
Sub Main()  
  If CheckBoxEnabled("Portrait") Then  
    SetCheckBox "Portrait",1  
  End If  
End Sub
```

See Also CheckBoxExists (function); GetCheckBox (function); SetCheckBox (statement).

Platform(s) Windows.

CheckBoxExists (function)

Syntax `CheckBoxExists(name$ | id)`

Description Returns True if the specified check box exists within the current window; returns False otherwise.

Comments The `CheckBoxExists` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the check box.
<i>id</i>	Integer specifying the ID of the check box.

Note: The `CheckBoxExists` function is used to determine whether a check box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.

Example 'This code fragment checks to ensure that the Portrait check box is 'selectable before selecting it.

```
Sub Main()
    If CheckBoxExists("Portrait") Then
        If CheckBoxEnabled("Portrait") Then
            SetCheckBox "Portrait",1
        End If
    End If
End Sub
```

See Also `CheckBoxEnabled (function)`; `GetCheckBox (function)`; `SetCheckBox (statement)`.

Platform(s) Windows.

Choose (function)

Syntax `Choose(index, expression1, expression2, . . . , expression13)`

Description Returns the expression at the specified index position.

Comments The *index* parameter specifies which expression is to be returned. If *index* is 1, then *expression1* is returned; if *index* is 2, then *expression2* is returned, and so on. If *index* is less than 1 or greater than the number of supplied expressions, then Null is returned.

The `Choose` function returns the expression without converting its type. Each expression is evaluated before returning the selected one.

Example 'This example assigns a variable of indeterminate type to a.

```
Sub Main()  
    Dim a As Variant  
    Dim c As Integer  
    c% = 2  
    a = Choose(c%,"Hello, world",#1/1/94#,5.5,False)  
    MsgBox "Item " & c% & " is '" & a & "'" 'Displays the date  
    passed as parameter 2.  
End Sub
```

See Also Switch (function); IIf (function); If...Then...Else (statement);
Select...Case (statement).

Platform(s) Windows and Macintosh.

Chr, Chr\$ (functions)

Syntax Chr[\$](*Code*)

Description Returns the character whose value is *Code*.

Comments *Code* must be an Integer between 0 and 255.

Chr\$ returns a string, whereas Chr returns a String variant.

The Chr\$ function can be used within constant declarations, as in the following example:

```
Const crlf = Chr$(13) + Chr$(10)
```

Some common uses of this function are:

Chr\$(9)	Tab
Chr\$(13) + Chr\$(10)	End-of-line (carriage return, linefeed)
Chr\$(26)	End-of-file
Chr\$(0)	Null

Example

```
Sub Main()  
    'Concatenates carriage return (13) and line feed (10) to CRLF$,  
    'then displays a multiple-line message using CRLF$ to separate  
    lines.  
    crlf$ = Chr$(13) + Chr$(10)  
    MsgBox "First line." & crlf$ & "Second line."  
  
    'Fills an array with the ASCII characters for ABC and displays  
    their  
    'corresponding characters.  
    Dim a%(2)  
    For i = 0 To 2  
        a%(i) = (65 + i)  
    Next i  
    MsgBox "The first three elements of the array are: " & Chr$(a%(0))  
    & Chr$(a%(1)) & Chr$(a%(2))  
End Sub
```

See Also Asc (function); Str, Str\$ (functions).

Platform(s) Windows and Macintosh.

CInt (function)

Syntax CInt (*expression*)

Description Converts *expression* to an Integer.

Comments This function accepts any expression convertible to an Integer, including strings. A runtime error is generated if *expression* is Null. Empty is treated as 0. The passed numeric expression must be within the valid range for integers:

$-32768 \leq \text{expression} \leq 32767$

A runtime error results if the passed expression is not within the above range.

When passed a numeric expression, this function has the same effect as assigning a numeric expression to an Integer. Note that integer variables are rounded before conversion.

When used with variants, this function guarantees that the expression is converted to an Integer variant (VarType 2).

Example 'This example demonstrates the various results of integer manipulation
'with CInt.

```
Sub Main()  
  
    '(1) Assigns i# to 100.55 and displays its integer representation  
(101).  
    i# = 100.55  
    MsgBox "The value of CInt(i) = " & CInt(i#)  
  
    '(2) Sets j# to 100.22 and displays the CInt representation (100).  
    j# = 100.22  
    MsgBox "The value of CInt(j) = " & CInt(j#)  
  
    '(3) Assigns k% (integer) to the CInt sum of j# and k% and displays  
k% (201).  
    k% = CInt(i# + j#)  
    MsgBox "The integer sum of 100.55 and 100.22 is: " & k%  
  
    '(4) Reassigns i# to 50.35 and recalculates k%, then displays the  
result  
    '(note rounding).  
    i# = 50.35  
    k% = CInt(i# + j#)  
    MsgBox "The integer sum of 50.35 and 100.22 is: " & k%  
  
End Sub
```

See Also CCur (function); CBool (function); CDate, CVDate (functions); CDb1
(function); CLng (function); CSng (function); CStr (function); CVar (function);
CVer (function); Integer (data type).

Platform(s) Windows and Macintosh.

Clipboard\$ (function)

Syntax Clipboard\$[()]

Description Returns a String containing the contents of the Clipboard.

Comments If the Clipboard doesn't contain text or the Clipboard is empty, then a zero-length string is returned.

Example 'This example puts text on the Clipboard, displays it, clears the
Clipboard,
'and displays the Clipboard again.

```
Const crlf = Chr$(13) + Chr$(10)  
  
Sub Main()  
    Clipboard$ "Hello out there!"  
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$  
    Clipboard.Clear  
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$  
End Sub
```

See Also Clipboard\$ (statement); Clipboard.GetText (method); Clipboard.SetText (method).

Platform(s) Windows and Macintosh.

Clipboard\$ (statement)

Syntax Clipboard\$ *NewContent\$*

Description Copies *NewContent\$* into the Clipboard.

Example 'This example puts text on the Clipboard, displays it, clears the Clipboard,
'and displays the Clipboard again.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Clipboard$ "Hello out there!"
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$
    Clipboard.Clear
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$
End Sub
```

See Also Clipboard\$ (function); Clipboard.GetText (method); Clipboard.SetText (method).

Platform(s) Windows and Macintosh.

Clipboard.Clear (method)

Syntax Clipboard.Clear

Description This method clears the Clipboard by removing any content.

Example 'This example puts text on the Clipboard, displays it, clears the Clipboard,
'and displays the Clipboard again.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Clipboard$ "Hello out there!"
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$
    Clipboard.Clear
    MsgBox "The text in the Clipboard is:" & crlf & Clipboard$
End Sub
```

Platform(s) Windows and Macintosh.

Clipboard.GetFormat (method)

Syntax WhichFormat = Clipboard.GetFormat(*format*)

Description Returns `True` if data of the specified format is available in the Clipboard; returns `False` otherwise.

Comments This method is used to determine whether the data in the Clipboard is of a particular format. The *format* parameter is an `Integer` representing the format to be queried:

Format	Description
1	Text
2	Bitmap
3	Metafile
8	Device-independent bitmap (DIB)
9	Color palette

Example 'This example puts text on the Clipboard, checks whether there is text on
'the Clipboard, and if there is, displays it.

```
Sub Main()
    Clipboard$ "Hello out there!"
    If Clipboard.GetFormat(1) Then
        MsgBox Clipboard$
    Else
        MsgBox "There is no text in the Clipboard."
    End If
End Sub
```

See Also Clipboard\$ (function); Clipboard\$ (statement).

Platform(s) Windows and Macintosh.

Clipboard.GetText (method)

Syntax `text$ = Clipboard.GetText([format])`

Description Returns the text contained in the Clipboard.

Comments The *format* parameter, if specified, must be 1.

Example 'This example retrieves the text from the Clipboard and checks to 'make sure that it contains the word "dog."

```
Option Compare Text

Sub Main()
    If Clipboard.GetFormat(1) Then
        If Instr(Clipboard.GetText(1),"dog",1) = 0 Then
            MsgBox "The Clipboard doesn't contain the word ""dog."" "
        Else
            MsgBox "The Clipboard contains the word ""dog"". "
        End If
    Else
        MsgBox "The Clipboard does not contain text."
    End If
End Sub
```

See Also Clipboard\$(statement); Clipboard\$(function); Clipboard.SetText (method).

Platform(s) Windows and Macintosh.

Clipboard.SetText (method)

Syntax Clipboard.SetText *data\$* [,*format*]

Description Copies the specified text string to the Clipboard.

Comments The *data\$* parameter specifies the text to be copied to the Clipboard. The *format* parameter, if specified, must be 1.

Example 'This example gets the contents of the Clipboard and uppercases it.

```
Sub Main()
    If Not Clipboard.GetFormat(1) Then Exit Sub
    Clipboard.SetText UCase$(Clipboard.GetText(1)),1
End Sub
```

See Also Clipboard\$(statement); Clipboard.GetText (method); Clipboard\$(function).

Platform(s) Windows and Macintosh.

CLng (function)

Syntax CLng(*expression*)

Description Converts *expression* to a Long.

Comments This function accepts any expression convertible to a Long, including strings. A runtime error is generated if *expression* is Null. Empty is treated as 0.

The passed expression must be within the following range:

-2147483648 <= *expression* <= 2147483647

A runtime error results if the passed expression is not within the above range.

When passed a numeric expression, this function has the same effect as assigning the numeric expression to a Long. Note that long variables are rounded before conversion.

When used with variants, this function guarantees that the expression is converted to a Long variant (VarType 3).

Example 'This example displays the results for various conversions of i and j (note 'rounding).

```
Sub Main()  
    i% = 100  
    j& = 123.666  
    MsgBox "The result is: " & CLng(i% * j&)           'Displays 12367.  
    MsgBox "The variant type is: " & Vartype(CLng(i%))  
End Sub
```

See Also CCur (function); CBool (function); CDate, CDate (functions); CDb1 (function); CInt (function); CSng (function); CStr (function); CVar (function); CVer (function); Long (data type).

Platform(s) Windows and Macintosh.

Close (statement)

Syntax Close [[#] *filename* [, [#] *filename*]...]

Description Closes the specified files.

Comments If no arguments are specified, then all files are closed.

Example 'This example opens four files and closes them in various combinations.

```
Sub Main()  
    Open "test1" For Output As #1  
    Open "test2" For Output As #2  
    Open "test3" For Random As #3  
    Open "test4" For Binary As #4  
    MsgBox "The next available file number is :" & FreeFile()  
    Close #1           'Closes file 1 only.  
    Close #2, #3       'Closes files 2 and 3.  
    Close              'Closes all remaining files(4).  
    MsgBox "The next available file number is :" & FreeFile()  
End Sub
```

See Also Open (statement); Reset (statement); End (statement).

Platform(s) Windows and Macintosh.

ComboBox (statement)

Syntax `ComboBox X,Y,width,height,ArrayVariable, .Identifier`

Description This statement defines a combo box within a dialog box template.

Comments When the dialog box is invoked, the combo box will be filled with the elements from the specified array variable.

This statement can only appear within a dialog box template (i.e., between the `Begin Dialog` and `End Dialog` statements).

The `ComboBox` statement requires the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>ArrayVariable</i>	Single-dimensioned array used to initialize the elements of the combo box. If this array has no dimensions, then the combo box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension.
	<i>ArrayVariable</i> can specify an array of any fundamental data type (structures are not allowed). <code>Null</code> and <code>Empty</code> values are treated as zero-length strings.
<i>.Identifier</i>	Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>). This parameter also creates a string variable whose value corresponds to the content of the edit field of the combo box. This variable can be accessed using the syntax:

DialogVariable.Identifier.

When the dialog box is invoked, the elements from *ArrayVariable* are placed into the combo box. The *.Identifier* variable defines the initial content of the edit field of the combo box. When the dialog box is dismissed, the *.Identifier* variable is updated to contain the current value of the edit field.

Example 'This example creates a dialog box that allows the user to select a day of the week.

```
Sub Main()
    Dim days$(6)
    days$(0) = "Monday"
    days$(1) = "Tuesday"
    days$(2) = "Wednesday"
    days$(3) = "Thursday"
    days$(4) = "Friday"
    days$(5) = "Saturday"
    days$(6) = "Sunday"

    Begin Dialog DaysDialogTemplate 16,32,124,96,"Days"
        OKButton 76,8,40,14,.OK
        Text 8,10,39,8,"&Weekdays:"
        ComboBox 8,20,60,72,days$,.Days
    End Dialog
    Dim DaysDialog As DaysDialogTemplate
    DaysDialog.Days = "Tuesday"
    r% = Dialog(DaysDialog)
    MsgBox "You selected: " & DaysDialog.Days
End Sub
```

See Also CancelButton (statement); CheckBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureBox (statement).

Platform(s) Windows and Macintosh.

ComboBoxEnabled (function)

Syntax ComboBoxEnabled(*name\$* | *id*)

Description Returns True if the specified combo box is enabled within the current window or dialog box; returns False otherwise.

Comments The ComboBoxEnabled function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the combo box. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the combo box. A runtime error is generated if the specified combo box does not exist.

Note: The ComboBoxEnabled function is used to determine whether a combo box is enabled in another application's dialog box. Use the DlgEnable function in dynamic dialog boxes.

Example 'This example checks to see whether a combo box is active. If it is, 'then it inserts some text into it.

```
Sub Main()
    If ComboBoxEnabled("Filename:") Then
        SelectComboBoxItem "Filename:", "sample.txt"
    End If
    If ComboBoxEnabled(365) Then
        SelectComboBoxItem 365,3      'Select the third item.
    End If
End Sub
```

See Also ComboBoxExists (function); GetComboBoxItem\$ (function);
GetComboBoxItemCount (function); SelectComboBoxItem (statement).

Platform(s) Windows.

ComboBoxExists (function)

Syntax ComboBoxExists(*name\$* | *id*)

Description Returns True if the specified combo box exists within the current window or dialog box; returns False otherwise.

Comments The `ComboBoxExists` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the combo box. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the combo box.
Note: The <code>ComboBoxExists</code> function is used to determine whether a combo box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example 'This code fragment checks to ensure that a combo box exists and is enabled
'before selecting the last item.

```
Sub Main()
    If ComboBoxExists("Filename:") Then
        If ComboBoxEnabled("Filename:") Then
            NumItems = GetComboBoxItemCount("Filename:")
            SelectComboBoxItem "Filename:",NumItems
        End If
    End If
End Sub
```

See Also `ComboBoxEnabled` (function); `GetComboBoxItem$` (function);
`GetComboBoxItemCount` (function); `SelectComboBoxItem` (statement).

Platform(s) Windows.

Command, Command\$ (functions)

Syntax `Command[$][()]`

Description Returns the argument from the command line used to start the application.

Comments `Command$` returns a string, whereas `Command` returns a `String` variant.

Example 'This example gets the command line and parameters, checks to see whether the string "/s" is present, and displays the result.

```
Sub Main()
    cmd$ = Command$
    If (InStr(cmd$,"/s")) <> 0 Then
        MsgBox "Application was started with the /s switch."
    Else
        MsgBox "Application was started without the /s switch."
    End If

    If cmd$ <> "" Then
        MsgBox "The command line startup options were: " & cmd$
    Else
        MsgBox "No command line startup options were used!"
    End If
End Sub
```

See Also Environ, Environ\$ (functions).

Platform(s) Windows and Macintosh.

Comments (topic)

Comments can be added to WM Basic code in the following manner:

All text between a single quotation mark and the end of the line is ignored:

```
MsgBox "Hello"           'Displays a message box.
```

The REM statement causes the compiler to ignore the entire line:

```
REM This is a comment.
```

WM Basic supports C-style multiline comment blocks /* . . . */, as shown in the following example:

```
MsgBox "Before comment"
/* This stuff is all commented out.
This line, too, will be ignored.
This is the last line of the comment. */
MsgBox "After comment"
```

Note: C-style comments can be nested.

Comparison Operators (topic)

Syntax *expression1* [< | > | <= | >= | <> | =] *expression2*

Description Comparison operators return True or False depending on the operator.

Comments The comparison operators are listed in the following table:

Operator	Returns True If
>	<i>expression1</i> is greater than <i>expression2</i>
<	<i>expression1</i> is less than <i>expression2</i>
<=	<i>expression1</i> is less than or equal to <i>expression2</i>
>=	<i>expression1</i> is greater than or equal to <i>expression2</i>
<>	<i>expression1</i> is not equal to <i>expression2</i>
=	<i>expression1</i> is equal to <i>expression2</i>

This operator behaves differently depending on the types of the expressions, as shown in the following table:

If one expression is	and the other expression is	then
Numeric below).	Numeric	A numeric comparison is performed (see below).
String below).	String	A string comparison is performed (see below).
Numeric	String	A compile error is generated.
Variant below).	String	A string comparison is performed (see below).
Variant below).	Numeric	A variant comparison is performed (see below).
Null variant	Any data type	Returns Null.
Variant below).	Variant	A variant comparison is performed (see below).

String Comparisons

If the two expressions are strings, then the operator performs a text comparison between the two string expressions, returning `True` if *expression1* is less than *expression2*. The text comparison is case-sensitive if `Option Compare` is `Binary`; otherwise, the comparison is case-insensitive.

When comparing letters with regard to case, lowercase characters in a string sort greater than uppercase characters, so a comparison of "a" and "A" would indicate that "a" is greater than "A".

Numeric Comparisons

When comparing two numeric expressions, the less precise expression is converted to be the same type as the more precise expression.

Dates are compared as doubles. This may produce unexpected results as it is possible to have two dates that, when viewed as text, display as the same date when, in fact, they are different. This can be seen in the following example:

```
Sub Main()
    Dim date1 As Date
    Dim date2 As Date

    date1 = Now
    date2 = date1 + 0.000001 'Adds a fraction of a second.

    MsgBox date2 = date1      'Prints False (the dates are
different).
    MsgBox date1 & "," & date2 'Prints two dates that are the
same.
End Sub
```

Variant Comparisons

When comparing variants, the actual operation performed is determined at execution time according to the following table:

If one variant is	and the other variant is	then
Numeric	Numeric	The variants are compared as numbers.
String	String	The variants are compared as text.
Numeric	String	The number is less than the string.
Null	Any other data type	Null.
Numeric	Empty	The number is compared with 0.
String string.	Empty	The string is compared with a zero-length

Example

```
Sub Main()  
    'Tests two literals and displays the result.  
    If 5 < 2 Then  
        MsgBox "5 is less than 2."  
    Else  
        MsgBox "5 is not less than 2."  
    End If  
  
    'Tests two strings and displays the result.  
    If "This" < "That" Then  
        MsgBox "'This' is less than 'That'."  
    Else  
        MsgBox "'That' is less than 'This'."  
    End If  
End Sub
```

See Also Operator Precedence (topic); Is (operator); Like (operator); Option Compare (statement).

Platform(s) Windows and Macintosh.

Const (statement)

Syntax `Const name [As type] = expression [, name [As type] = expression]...`

Description Declares a constant for use within the current script.

Comments The *name* is only valid within the current WM Basic script. Constant names must follow these rules:

1. Must begin with a letter.
2. May contain only letters, digits, and the underscore character.
3. Must not exceed 80 characters in length.
4. Cannot be a reserved word.

Constant names are not case-sensitive.

The *expression* must be assembled from literals or other constants. Calls to functions are not allowed except calls to the Chr\$ function, as shown below:

```
Const s$ = "Hello, there" + Chr(44)
```

Constants can be given an explicit type by declaring the *name* with a type-declaration character, as shown below:

```
Const a% = 5           'Constant Integer whose value is 5  
Const b# = 5           'Constant Double whose value is 5.0  
Const c$ = "5"         'Constant String whose value is "5"  
Const d! = 5           'Constant Single whose value is 5.0  
Const e& = 5           'Constant Long whose value is 5
```

The type can also be given by specifying the *As type* clause:

```
Const a As Integer = 5      'Constant Integer whose value is 5
Const b As Double = 5       'Constant Double whose value is 5.0
Const c As String = "5"     'Constant String whose value is "5"
Const d As Single = 5       'Constant Single whose value is 5.0
Const e As Long = 5         'Constant Long whose value is 5
```

You cannot specify both a type-declaration character and the *type*:

```
Const a% As Integer = 5     'THIS IS ILLEGAL.
```

If an explicit type is not given, then WM Basic will choose the most imprecise type that completely represents the data, as shown below:

```
Const a = 5                'Integer constant
Const b = 5.5              'Single constant
Const c = 5.5E200          'Double constant
```

Constants defined within a Sub or Function are local to that subroutine or function. Constants defined outside of all subroutines and function can be used anywhere within that script. The following example demonstrates the scoping of constants:

```
Const DefFile = "default.txt"

Sub Test1
    Const DefFile = "foobar.txt"
    MsgBox DefFile           'Displays "foobar.txt".
End Sub

Sub Test2
    MsgBox DefFile           'Displays
"default.txt".
End Sub
```

Example 'This example displays the declared constants in a dialog box (crlf produces
'a new line in the dialog box).

```
Const crlf = Chr$(13) + Chr$(10)
Const s As String = "This is a constant."

Sub Main()
    MsgBox s$ & crlf & "The constants are shown above."
End Sub
```

See Also DefType (statement); Let (statement); = (statement); Constants (topic).
Platform(s) Windows and Macintosh.

Constants (topic)

Constants are variables that cannot change value during script execution. The following constants are predefined by WM Basic:

True	False	Empty
Pi	ebRightButton	ebLeftButton
ebPortrait	ebLandscape	ebEmpty
ebWindows	ebMaximized	ebMinimized
ebRestored	ebNormal	ebReadOnly
ebHidden	ebSystem	ebVolume
ebDirectory	ebArchive	ebNone
ebOKOnly	ebOKCancel	ebAbortRetryIgnore
ebYesNoCancel	ebYesNo	ebRetryCancel
ebCritical	ebQuestion	ebExclamation
ebInformation	ebApplicationModal	
ebDefaultButton1		
ebDefaultButton2	ebDefaultButton3	ebSystemModal
ebOK	ebCancel	ebAbort
ebRetry	ebIgnore	ebYes
ebNo	ebWin16	ebMacintosh
ebInteger	ebLong	ebSingle
ebDouble	ebDate	ebBoolean
ebObject	ebDataObject	ebVariant
ebCurrency	ebNull	

You can define your own constants using the Const statement.

Cos (function)

Syntax `Cos(angle)`

Description Returns a Double representing the cosine of *angle*.

Comments The *angle* parameter is a Double specifying an angle in radians.

Example 'This example assigns the cosine of pi/4 radians (45 degrees) to C# and displays its value.

```
Sub Main()
    c# = Cos(3.14159 / 4)
    MsgBox "The cosine of 45 degrees is: " & c#
End Sub
```

See Also Tan (function); Sin (function); Atn (function).

Platform(s) Windows and Macintosh.

CreateObject (function)

Syntax

```
CreateObject(class$)
```

Description Creates an OLE automation object and returns a reference to that object.

Comments The *class\$* parameter specifies the application used to create the object and the type of object being created. It uses the following syntax:

"*application.class*",

where *application* is the application used to create the object and *class* is the type of the object to create.

At runtime, `CreateObject` looks for the given application and runs that application if found. Once the object is created, its properties and methods can be accessed using the dot syntax (e.g., `object.property = value`).

There may be a slight delay when an automation server is loaded (this depends on the speed with which a server can be loaded from disk). This delay is reduced if an instance of the automation server is already loaded.

Examples

'This first example instantiates Microsoft Excel. It then uses the resulting object to make Excel visible and then close Excel.

[illegible]

```
'This second example uses CreateObject to instantiate a Visio object.
It
'then uses the resulting object to create a new document.

Sub Main()
    Dim Visio As Object
    Dim doc As Object
    Dim page As Object
    Dim shape As Object

    Set Visio = CreateObject("visio.application") 'Create Visio object.
    Set doc = Visio.Documents.Add("") 'Create a new document.
    Set page = doc.Pages(1) 'Get first page.
    Set shape = page.DrawRectangle(1,1,4,4) 'Create a new shape.
    shape.text = "Hello, world." 'Set text within
shape.
End Sub
```

See Also GetObject (function); Object (data type).

Platform(s) Windows and Macintosh..

Cross-Platform Scripting (topic)

This section discusses different techniques that can be used to ensure that a given script runs on all platforms that support WM Basic.

Querying the Platform

A script can query the platform in order to take appropriate actions for that platform. This is done using the `Basic.OS` property. The following example uses this method to display a message to the user:

```
Sub Main()
    If Basic.OS = ebWin16 Then
        MsgBox "Running on Windows"
    Else
        Print "Not running on Windows"
    End If
End Sub
```

Querying the Capabilities of a Platform

Some capabilities of the current platform can be determined using the `Basic.Capability` method. This method takes a number indicating which capability is being queried and returns either `True` or `False` depending on whether that capability is or is not supported on the current platform. The following example uses this technique to read hidden files:

```
Sub Main()  
  If Basic.Capability(3) Then  
    f$ = Dir$("*",ebHidden)'This platform supports hidden files.  
  Else  
    f$ = Dir$("*")           'This platform doesn't support  
hidden files.  
  End If  
  
  'Print all the files.  
  While f$ <> ""  
    x = x + 1  
    MsgBox "Matching file " & x & " is: " & f$  
    f$ = Dir$  
  WEnd  
End Sub
```

Byte Ordering with Files

One of the main differences between platforms is byte ordering. On some platforms, the processor requires that the bytes that make up a given data item be reversed from their expected ordering.

Byte ordering becomes problematic if binary data is transferred from one platform to another. This can only occur when writing data to files. For this reason, it is strongly recommended that files that are to be transported to a different platform with different byte ordering be sequential (i.e., do not use `Binary` and `Random` files).

If a `Binary` or `Random` file needs to be transported to another platform, you will have to take into consideration the following:

1. You must either decide on a byte ordering for your file or write information to the file indicating its byte ordering so that it can be queried by the script that is to read the file.
2. When reading a file on a platform in which the byte ordering matches, nothing further needs to be done. If the byte ordering is different, then the bytes of each data item read from a file need to be reversed. This is a difficult proposition.

Byte Ordering with Structures

Due to byte ordering differences between platforms, structure copying using the `LSet` statement produces different results. Consider the following example:

```
Type TwoInts
    first As Integer
    second As Integer
End Type

Type OneLong
    first As Long
End Type

Sub Main()
    Dim l As OneLong
    Dim i As TwoInts
    l.First = 4
    LSet i = l
    MsgBox "First integer: " & i.first
    MsgBox "Second integer: " & i.second
End Sub
```

On Intel-based platforms, bytes are stored in memory with the most significant byte first (known as little-endian format). Thus, the above example displays two dialog boxes, the first one displaying the number 4 and the second displaying the number 0.

On Macintosh platforms, bytes are stored in memory with the least significant byte first (known as big-endian format). Thus, the above example displays two dialog boxes, the first one displaying the number 0 and the second displaying the number 4.

Script that rely on binary images of data must take the byte ordering of the current platform into account.

Reading Text Files and Writing to Them

Different platforms use different characters to represent end-of-line in a file. For example, under Windows, a carriage-return/line-feed pair is used. On the Macintosh, a carriage return is used.

WM Basic takes this into account when reading text files. The following combinations are recognized and interpreted as end-of-line:

Carriage return	<code>Chr(13)</code>
Carriage return/line feed	<code>Chr(13) + Chr(10)</code>
Line feed	<code>Chr(10)</code>

When writing to text files, WM Basic uses the end-of-line appropriate to that platform. You can retrieve the same end-of-line used by WM Basic using the `Basic.Eoln$` property:

```
crlf = Basic.Eoln$
Print #1,"Line 1." & crlf & "Line 2."
```

Alignment

A major difference between platforms supported by WM Basic is the forced alignment of data. WM Basic handles most alignment issues itself.

Portability of Compiled Code

Scripts compiled under WM Basic can be executed without recompilation on any platform supported by WM Basic.

Unsupported Language Elements

A compiled WM Basic script is portable to any platform on which WM Basic runs. Because of this, it is possible to execute a script that was compiled on another platform and contains calls to language elements not supported by the current platform.

WM Basic generates a runtime error when unsupported language elements are encountered during execution.

If you trap a call to an unsupported function, the function will return one of the following values:

Data Type	Skipped Function Returns
Integer	0
Double	0.0
Single	0.0
Long	0
Date	December 31, 1899
Boolean	False
Variant	Empty
Object	Nothing

Path Separators

Different file systems use different characters to separate parts of a pathname. For example, under Windows, the backslash character is used:

```
s$ = "c:\sheets\bob.xls"
```

When creating scripts that operate on any of these platforms, WM Basic recognizes the forward slash universally as a valid path separator. Thus, the following file specification is valid on all these platforms:

```
s$ = "/sheets/bob.xls"
```

On the Macintosh, the slashes are valid filename characters. Instead, WM Basic recognizes the colon as the valid file separator character:

```
s$ = "sheets:bob.xls"
```

You can find out the path separator character for your platform using the `Basic.PathSeparator$` property:

```
s$ = "sheets" & Basic.PathSeparator$ & "bob.xls"
```

Relative Paths

Specifying relative paths is different across platforms. Under Windows, a period (.) is used to specify the current directory, and two periods (..) are used to indicate the parent directory, as shown below:

```
s$ = ".\bob.xls"      'File in the current directory
s$ = "..\bob.xls"     'File in the parent directory
```

On the Macintosh, double colons are used to specify the parent folder:

```
s$ = "::bob.xls"      'File in the parent folder
```

Drive Letters

Not all platforms support drive letters. For example, considering the following file specification:

```
c:\test.txt
```

Under Windows, this specifies a file called `test.txt` in the root directory of drive `c`. On the Macintosh, this specifies a file called `\test.txt` in a folder called `c`. You can use the `Basic.Capability` method to determine whether your platform supports drive letters:

```
Sub Main()
  If Basic.Capability(1) Then s$ = "c:/" Else s$ = ""
  s$ = s$ & "test.xls"
  MsgBox "The platform-specific filename is: " & s$
End Sub
```

CSng (function)

Syntax `CSng (expression)`

Description Converts *expression* to a `Single`.

Comments This function accepts any expression convertible to a `Single`, including strings. A runtime error is generated if *expression* is `Null`. Empty is treated as `0.0`.

A runtime error results if the passed expression is not within the valid range for `Single`.

When passed a numeric expression, this function has the same effect as assigning the numeric expression to a `Single`.

When used with variants, this function guarantees that the expression is converted to a `Single` variant (`VarType 4`).

Example 'This example displays the value of a String converted to a Single.

```
Sub Main()
    s$ = "100"
    MsgBox "The single value is: " & CSng(s$)
End Sub
```

See Also CCur (function); CBool (function); CDate, CDate (functions); CDb1 (function); CInt (function); CLng (function); CStr (function); CVar (function); CVer (function); Single (data type).

Platform(s) Windows and Macintosh.

CStr (function)

Syntax CStr(*expression*)

Description Converts *expression* to a String.

Comments Unlike Str\$ or Str, the string returned by CStr will not contain a leading space if the expression is positive. Further, the CStr function correctly recognizes thousands and decimal separators for your locale.

Different data types are converted to String in accordance with the following rules:

Data Type	CStr Returns
Any numeric type	A string containing the number without the leading space for positive values.
Date	A string converted to a date using the short date format.
Boolean	A string containing either "True" or "False".
Null variant	A runtime error.
Empty variant	A zero-length string.

Example 'This example displays the value of a Double converted to a String. Sub Main()
 s# = 123.456
 MsgBox "The string value is: " & CStr(s#)
 End Sub

See Also CCur (function); CBool (function); CDate, CDate (functions); CDb1 (function); CInt (function); CLng (function); CSng (function); CVar (function); CVer (function); String (data type); Str, Str\$ (functions).

Platform(s) Windows and Macintosh.

CurDir, CurDir\$ (functions)

Syntax CurDir[\$][(drive\$)]

Description Returns the current directory on the specified drive. If no *drive\$* is specified or *drive\$* is zero-length, then the current directory on the current drive is returned.

Comments CurDir\$ returns a String, whereas CurDir returns a String variant.

WM Basic generates a runtime error if *drive\$* is invalid.

Example 'This example saves the current directory, changes to the next higher directory, and displays the change; then restores the original directory and displays the change. Note: The dot designators will not work with all platforms.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    save$ = CurDir$
    ChDir ("..")
    MsgBox "Old directory: " & save$ & crlf & "New directory: " &
CurDir$
    ChDir (save$)
    MsgBox "Directory restored to: " & CurDir$
End Sub
```

See Also ChDir (statement); ChDrive (statement); Dir, Dir\$ (functions); Mkdir (statement); Rmdir (statement).

Platform(s) Windows and Macintosh.

Currency (data type)

Syntax Currency

Description A data type used to declare variables capable of holding fixed-point numbers with 15 digits to the left of the decimal point and 4 digits to the right.

Comments Currency variables are used to hold numbers within the following range:

-922,337,203,685,477.5808 <= currency <= 922,337,203,685,477.5807

Due to their accuracy, Currency variables are useful within calculations involving money.

The type-declaration character for Currency is @.

Storage

Internally, currency values are 8-byte integers scaled by 10000. Thus, when appearing within a structure, currency values require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required.

See Also Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CCur (function).

Platform(s) Windows and Macintosh.

CVar (function)

Syntax `CVar (expression)`

Description Converts *expression* to a Variant.

Comments This function is used to convert an expression into a variant. Use of this function is not necessary (except for code documentation purposes) because assignment to variant variables automatically performs the necessary conversion:

```
Sub Main()
    Dim v As Variant
    v = 4 & "th"           'Assigns "4th" to v.
    MsgBox "You came in: " & v
    v = CVar(4 & "th")     'Assigns "4th" to v.
    MsgBox "You came in: " & v
End Sub
```

Example 'This example converts an expression into a Variant.

```
Sub Main()
    Dim s As String
    Dim a As Variant
    s = CStr("The quick brown fox ")
    msg = CVar(s & "jumped over the lazy dog.")
    MsgBox msg
End Sub
```

See Also CCur (function); CBool (function); CDate, CDate (functions); CDb1 (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function); Variant (data type).

Platform(s) Windows and Macintosh.

CVar (function)

Syntax `CVar (expression)`

Description Converts *expression* to an error.

Comments This function is used to convert an expression into a user-defined error number. A runtime error is generated under the following conditions:

- If *expression* is Null.
- If *expression* is a number outside the legal range for errors, which is as follows:

`0 <= expression <= 65535`

- If *expression* is Boolean.
- If *expression* is a String that can't be converted to a number within the legal range.

Empty is treated as 0.

Example 'This example simulates a user-defined error and displays the error number.

```
Sub Main()  
    MsgBox "The error is: " & CStr(CVErr(2046))  
End Sub
```

See Also CCur (function); CBool (function); CDate, CVDate (functions); CDb1 (function); CInt (function); CLng (function); CSng (function); CStr (function); CVar (function), IsError (function).

Platform(s) Windows and Macintosh.

Date (data type)

Syntax `Date`

Description A data type capable of holding date and time values.

Comments Date variables are used to hold dates within the following range:

```
January 1, 100 00:00:00 <= date <= December 31, 9999 23:59:59
-6574340 <= date <= 2958465.99998843
```

Internally, dates are stored as 8-byte IEEE double values. The integer part holds the number of days since December 31, 1899, and the fractional part holds the number of seconds as a fraction of the day. For example, the number 32874.5 represents January 1, 1990 at 12:00:00.

When appearing within a structure, dates require 8 bytes of storage. Similarly, when used with binary or random files, 8 bytes of storage are required.

There is no type-declaration character for `Date`.

Date variables that haven't been assigned are given an initial value of 0 (i.e., December 31, 1899).

Date Literals

Literal dates are specified using number signs, as shown below:

```
Dim d As Date
d = #January 1, 1990#
```

The interpretation of the date string (i.e., January 1, 1990 in the above example) occurs at runtime, using the current country settings. This is a problem when interpreting dates such as 1/2/1990. If the date format is M/D/Y, then this date is January 2, 1990. If the date format is D/M/Y, then this date is February 1, 1990. To remove any ambiguity when interpreting dates, use the universal date format:

```
date_variable = #YY/MM/DD HH:MM:SS#
```

The following example specifies the date June 3, 1965 using the universal date format:

```
Dim d As Date
d = #1965/6/3 10:23:45#
```

See Also `Currency (data type)`; `Double (data type)`; `Integer (data type)`; `Long (data type)`; `Object (data type)`; `Single (data type)`; `String (data type)`; `Variant (data type)`; `Boolean (data type)`; `DefType (statement)`; `CDate`, `CVDate` (functions).

Platform(s) Windows and Macintosh.

Date, Date\$ (functions)

Syntax Date[\$][()]

Description Returns the current system date.

Comments The Date\$ function returns the date using the short date format. The Date function returns the date as a Date variant.

Use the Date/Date\$ statements to set the system date.

Note: In prior versions of WM Basic, the Date\$ function returned the date using a fixed date format. The date is now returned using the current short date format (defined by the operating system), which may differ from the previous fixed format.

Example 'This example saves the current date to Cdate\$, then changes the 'date and displays the result. It then changes the date back to the 'saved date and displays the result.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    TheDate$ = Date$()
    Date$ = "01/01/95"
    MsgBox "Saved date is: " & TheDate$ & crlf & "Changed date is: " &
Date$()
    Date$ = TheDate$
    MsgBox "Restored date to: " & TheDate$
End Sub
```

See Also CDate, CVDDate (functions); Time, Time\$ (functions); Date, Date\$ (statements); Now (function); Format, Format\$ (functions); DateSerial (function); DateValue (function).

Platform(s) Windows and Macintosh.

Date, Date\$ (statements)

Syntax Date[\$] = *newdate*

Description Sets the system date to the specified date.

Comments The Date\$ statement requires a string variable using one of the following formats:

```
MM-DD-YYYY
MM-DD-YY
MM/DD/YYYY
MM/DD/YY,
```

where *MM* is a two-digit month between 1 and 31, *DD* is a two-digit day between 1 and 31, and *YYYY* is a four-digit year between 1/1/100 and 12/31/9999.

The Date statement converts any expression to a date, including string and numeric values. Unlike the Date\$ statement, Date recognizes many different date formats, including abbreviated and full month names and a variety of ordering options. If *newdate* contains a time component, it is accepted, but the time is not changed. An error occurs if *newdate* cannot be interpreted as a valid date.

Example 'This example saves the current date to Cdate\$, then changes the 'date and displays the result. It then changes the date back to the 'saved date and displays the result.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    TheDate$ = Date$()
    Date$ = "01/01/95"
    MsgBox "Saved date is: " & TheDate$ & crlf & "Changed date is: " &
Date$()
    Date$ = TheDate$
    MsgBox "Restored date to: " & TheDate$
End Sub
```

See Also Date, Date\$ (functions); Time, Time\$ (statements).

Platform(s) Windows and Macintosh.

Platform Notes On some platforms, you may not have permission to change the date, causing runtime error 70 to be generated. This can occur on Win32 and OS/2.

The range of valid dates varies from platform to platform. The following table describes the minimum and maximum dates accepted by various platforms:

Platform	Minimum Date	Maximum Date
Macintosh	January 1, 1904	February 6, 2040
Windows	January 1, 1980	December 31, 2099
Windows 95	January 1, 1980	December 31, 2099
OS/2	January 1, 1980	December 31, 2079

DateAdd (function)

Syntax `DateAdd(interval$, increment&, date)`

Description Returns a `Date` variant representing the sum of *date* and a specified number (*increment*) of time intervals (*interval\$*).

Comments This function adds a specified number (*increment*) of time intervals (*interval\$*) to the specified date (*date*). The following table describes the parameters to the `DateAdd` function:

Parameter	Description
<i>interval\$</i>	String expression indicating the time interval used in the addition.
<i>increment</i>	Integer indicating the number of time intervals you wish to add. Positive values result in dates in the future; negative values result in dates in the past.
<i>date</i>	Any expression convertible to a <code>Date</code> .
	The <i>interval\$</i> parameter specifies what unit of time is to be added to the given date. It can be any of the following:
Time	Interval
"Y"	Day of the year
"YYYY"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

To add days to a date, you may use either day, day of the year, or weekday, as they are all equivalent ("d", "Y", "w").

The `DateAdd` function will never return an invalid date/time expression. The following example adds two months to December 31, 1992:

```
s# = DateAdd("m", 2, "December 31, 1992")
```

In this example, `s` is returned as the double-precision number equal to "February 28, 1993", not "February 31, 1993".

WM Basic generates a runtime error if you try subtracting a time interval that is larger than the time value of the date.

Example 'This example gets today's date using the Date\$ function; adds three years, two months, one week, and two days to it; and then displays the result in a dialog box.

```
Sub Main()
    Dim sdate$
    sdate$ = Date$
    NewDate# = DateAdd("yyyy", 4, sdate$)
    NewDate# = DateAdd("m", 3, NewDate#)
    NewDate# = DateAdd("ww", 2, NewDate#)
    NewDate# = DateAdd("d", 1, NewDate#)
    s$ = "Four years, three months, two weeks, and one day from now
will be: "
    s$ = s$ & Format(NewDate#, "long date")
    MsgBox s$
End Sub
```

See Also DateDiff (function).

Platform(s) Windows and Macintosh.

DateDiff (function)

Syntax DateDiff(*interval\$,date1 ,date2*)

Description Returns a Date variant representing the number of given time intervals between *date1* and *date2*.

Comments The following table describes the parameters:

Parameter	Description
<i>interval\$</i>	String expression indicating the specific time interval you wish to find the difference between.
<i>date1</i>	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1994".
<i>date2</i>	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1994".

The following table lists the valid time interval strings and the meanings of each. The `Format$` function uses the same expressions.

Time	Interval
"Y"	Day of the year
"YYYY"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

To find the number of days between two dates, you may use either `day` or `day of the year`, as they are both equivalent ("d", "Y").

The time interval `weekday` ("w") will return the number of weekdays occurring between *date1* and *date2*, counting the first occurrence but not the last. However, if the time interval is `week` ("ww"), the function will return the number of calendar weeks between *date1* and *date2*, counting the number of Sundays. If *date1* falls on a Sunday, then that day is counted, but if *date2* falls on a Sunday, it is not counted.

The `DateDiff` function will return a negative date/time value if *date1* is a date later in time than *date2*.

Example 'This example gets today's date and adds ten days to it. It then
'calculates the difference between the two dates in days and weeks
'and displays the result.

```
Sub Main()  
    today$ = Format(Date$, "Short Date")  
    NextWeek = Format(DateAdd("d", 14, today$), "Short Date")  
    DifDays# = DateDiff("d", today$, NextWeek)  
    DifWeek# = DateDiff("w", today$, NextWeek)  
    s$ = "The difference between " & today$ & " and " & NextWeek  
    s$ = s$ & " is: " & DifDays# & " days or " & DifWeek# & " weeks"  
    MsgBox s$  
End Sub
```

See Also `DateAdd` (function).

Platform(s) Windows and Macintosh.

DatePart (function)

Syntax `DatePart(interval$, date)`

Description Returns an Integer representing a specific part of a date/time expression.

Comments The DatePart function decomposes the specified date and returns a given date/time element. The following table describes the parameters:

Parameter	Description
<i>interval\$</i>	String expression that indicates the specific time interval you wish to identify within the given date.
<i>date</i>	Any expression convertible to a Date. An example of a valid date/time string would be "January 1, 1995". The following table lists the valid time interval strings and the meanings of each. The Format\$ function uses the same expressions.
Time	Interval
"Y"	Day of the year
"YYYY"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday

The weekday expression starts with Sunday as 1 and ends with Saturday as 7.

Example 'This example displays the parts of the current date.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    today$ = Date$
    qtr = DatePart("q",today$)
    yr = DatePart("yyyy",today$)
    mo = DatePart("m",today$)
    wk = DatePart("ww",today$)
    da = DatePart("d",today$)
    s$ = "Quarter: " & qtr & crlf
    s$ = s$ & "Year      : " & yr & crlf
    s$ = s$ & "Month     : " & mo & crlf
    s$ = s$ & "Week      : " & wk & crlf
    s$ = s$ & "Day       : " & da & crlf
    MsgBox s$
End Sub
```

See Also Day (function); Minute (function); Second (function); Month (function); Year (function); Hour (function); Weekday (function), Format (function).

Platform(s) Windows and Macintosh.

DateSerial (function)

Syntax DateSerial(*year*,*month*,*day*)

Description Returns a Date variant representing the specified date.

Comments The DateSerial function takes the following parameters:

Parameter	Description
<i>year</i>	Integer between 100 and 9999
<i>month</i>	Integer between 1 and 12
<i>day</i>	Integer between 1 and 31

Example 'This example converts a date to a real number representing the 'serial date in days since December 30, 1899 (which is day 0).

```
Sub Main()
    tdate# = DateSerial(1993,08,22)
    MsgBox "The DateSerial value for August 22, 1993, is: " & tdate#
End Sub
```

See Also DateValue (function); TimeSerial (function); TimeValue (function); CDate, CVDat (functions).

Platform(s) Windows and Macintosh.

DateValue (function)

Syntax DateValue(*date_string*)

Description Returns a Date variant representing the date contained in the specified string argument.

Example 'This example returns the day of the month for today's date.

```
Sub Main()
    tdate$ = Date$
    tday = DateValue(tdate$)
    MsgBox tdate & " date value is: " & tday$
End Sub
```

See Also TimeSerial (function); TimeValue (function); DateSerial (function).

Platform(s) Windows and Macintosh.

Platform Notes: Under Windows, date specifications vary depending on the international settings contained in the "intl" section of the win.ini file. The date items must follow the ordering determined by the current date format settings in use by Windows.

Day (function)

Syntax Day(*date*)

Description Returns the day of the month specified by *date*.

Comments The value returned is an Integer between 0 and 31 inclusive.

The *date* parameter is any expression that converts to a Date.

Example 'This example gets the current date and then displays it.

```
Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()
```

```
    CurDate = Now()
```

```
    MsgBox "Today is day " & Day(CurDate) & " of the month." & crlf &  
    "Tomorrow is day " & Day(CurDate + 1)
```

```
End Sub
```

See Also Minute (function); Second (function); Month (function); Year (function); Hour (function); Weekday (function); DatePart (function).

Platform(s) Windows and Macintosh.

DDB (function)

Syntax DDB(*Cost, Salvage, Life, Period*)

Description Calculates the depreciation of an asset for a specified *Period* of time using the double-declining balance method.

Comments The double-declining balance method calculates the depreciation of an asset at an accelerated rate. The depreciation is at its highest in the first period and becomes progressively lower in each additional period. DDB uses the following formula to calculate the depreciation:

$$\text{DDB} = ((\text{Cost} - \text{Total_depreciation_from_all_other_periods}) * 2) / \text{Life}$$

The DDB function uses the following parameters:

Parameter	Description
<i>Cost</i>	Double representing the initial cost of the asset
<i>Salvage</i>	Double representing the estimated value of the asset at the end of its predicted useful life
<i>Life</i>	Double representing the predicted length of the asset's useful life
<i>Period</i>	Double representing the period for which you wish to calculate the depreciation

Life and *Period* must be expressed using the same units. For example, if *Life* is expressed in months, then *Period* must also be expressed in months.

Example 'This example calculates the depreciation for capital equipment
'that cost \$10,000, has a service life of ten years, and is worth
'\$2,000 as scrap. The dialog box displays the depreciation for each
'of the first four years.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    s$ = "Depreciation Table" & crlf & crlf
    For yy = 1 To 4
        CurDep# = DDB(10000.0,2000.0,10,yy)
        s$ = s$ & "Year " & yy & " : " & CurDep# & crlf
    Next yy
    MsgBox s$
End Sub
```

See Also Sln (function); SYD (function).

Platform(s) Windows and Macintosh.

DDEExecute (statement)

Syntax DDEExecute *channel, command\$*

Description Executes a command in another application.

Comments The DDEExecute statement takes the following parameters:

Parameter	Description
-----------	-------------

<i>channel</i>	Integer containing the DDE channel number returned from DDEInitiate. An error will result if <i>channel</i> is invalid.
----------------	---

<i>command\$</i>	String containing the command to be executed. The format of <i>command\$</i> depends on the receiving application.
------------------	--

If the receiving application does not execute the instructions, WM Basic generates a runtime error.

Example 'This example selects a cell in an Excel spreadsheet.

```
Sub Main()
    q$ = Chr(34)
    ch% = DDEInitiate("Excel", "c:\sheets\test.xls")
    cmd$ = "Select(" & q$ & "R1C1:R8C1" & q$ & ")"
    DDEExecute ch%, cmd$
    DDETerminate ch%
End Sub
```

See Also DDEInitiate (function); DDEPoke (statement); DDERequest, DDERequest\$ (functions); DDESend (function); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Platform(s) Windows.

Platform Notes: Under Windows, the DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the WM Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

DDEInitiate (function)

Syntax DDEInitiate(application\$, topic\$)

Description Initializes a DDE link to another application and returns a unique number subsequently used to refer to the open DDE channel.

Comments The `DDEInitiate` statement takes the following parameters:

Parameter	Description
<i>application\$</i>	String containing the name of the application (the server) with which a DDE conversation will be established.
<i>topic\$</i>	<p>String containing the name of the topic for the conversation. The possible values for this parameter are described in the documentation for the server application.</p> <p>This function returns 0 if WM Basic cannot establish the link. This will occur under any of the following circumstances:</p> <ul style="list-style-type: none"> • The specified application is not running. • The topic was invalid for that application. • Memory or system resources are insufficient to establish the DDE link.

Example 'This example selects a range of cells in an Excel spreadsheet.

```
Sub Main()
    q$ = Chr(34)
    ch% = DDEInitiate("Excel", "c:\sheets\test.xls")
    cmd$ = "Select(" & q$ & "R1C1:R8C1" & q$ & ")"
    DDEExecute ch%, cmd$
    DDETerminate ch%
End Sub
```

See Also `DDEExecute (statement)`; `DDEPoke (statement)`; `DDERequest, DDERequest$ (functions)`; `DDESend (function)`; `DDETerminate (statement)`; `DDETerminateAll (statement)`; `DDETimeout (statement)`.

Platform(s) Windows.

Platform Notes: Under Windows, the DDEML library is required for DDE support. This library is loaded when the first `DDEInitiate` statement is encountered and remains loaded until the WM Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

DDEPoke (statement)

Syntax `DDEPoke channel, DataItem, value`

Description Sets the value of a data item in the receiving application associated with an open DDE link.

Comments The DDEPoke statement takes the following parameters:

Parameter	Description
<i>channel</i>	Integer containing the DDE channel number returned from DDEInitiate. An error will result if <i>channel</i> is invalid.
<i>DataItem</i>	Data item to be set. This parameter can be any expression convertible to a String. The format depends on the server.
<i>value</i>	The new value for the data item. This parameter can be any expression convertible to a String. The format depends on the server. A runtime error is generated if <i>value</i> is Null.

Example 'This example pokes a value into an Excel spreadsheet.

```
Sub Main()
    ch% = DDEInitiate("Excel", "c:\sheets\test.xls")
    DDEPoke ch%, "R1C1", "980"
    DDETerminate ch%
End Sub
```

See Also DDEExecute (statement); DDEInitiate (function); DDERequest, DDERequest\$ (functions); DDESend (function); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Platform(s) Windows.

Platform Notes: Under Windows, the DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the WM Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

DDERequest, DDERequest\$ (functions)

Syntax DDERequest[\$] (*channel*, *DataItem*\$)

Description Returns the value of the given data item in the receiving application associated with the open DDE channel.

Comments DDERequest\$ returns a String, whereas DDERequest returns a String variant.

The DDERequest/DDERequest\$ functions take the following parameters:

Parameter	Description
<i>channel</i>	Integer containing the DDE channel number returned from DDEInitiate. An error will result if <i>channel</i> is invalid.
<i>DataItem\$</i>	String containing the name of the data item to request. The format for this parameter depends on the server.

The format for the returned value depends on the server.

Example 'This example gets a value from an Excel spreadsheet.

```
Sub Main()
    ch% = DDEInitiate("Excel", "c:\excel\test.xls")
    s$ = DDERequest$(ch%, "R1C1")
    DDETerminate ch%
    MsgBox s$
End Sub
```

See Also DDEExecute (statement); DDEInitiate (function); DDEPoke (statement); DDESend (function); DDETerminate (statement); DDETerminateAll (statement); DDETimeout (statement).

Platform(s) Windows.

Platform Notes: Under Windows, the DDEML library is required for DDE support. This library is loaded when the first DDEInitiate statement is encountered and remains loaded until the WM Basic system is terminated. Thus, the DDEML library is required only if DDE statements are used within a script.

DDESend (statement)

Syntax DDESend *application\$, topic\$, DataItem, value*

Description Initiates a DDE conversation with the server as specified by *application\$* and *topic\$* and sends that server a new value for the specified item.

The **DDETerminate** statement takes the following parameters:

Parameter	Description
<i>application\$</i>	String containing the name of the application (the server) with which a DDE conversation will be established.
<i>topic\$</i>	String containing the name of the topic for the conversation. The possible values for this parameter are described in the documentation for the server application.
<i>DataItem</i>	Data item to be set. This parameter can be any expression convertible to a String. The format depends on the server.
<i>value</i>	New value for the data item. This parameter can be any expression convertible to a String. The format depends on the server. A runtime error is generated if <i>value</i> is Null.

The **DDESend** statement performs the equivalent of the following statements:

```
ch% = DDEInitiate(application$, topic$)
DDEPoke ch%, item, data
DDETerminate ch%
```

Example

```
'This code fragment sets the content of the first cell in
'an Excel spreadsheet.

Sub Main()
    On Error Goto Trap1
    DDESend "Excel", "c:\excel\test.xls", "R1C1", "Hello, world."
    On Error Goto 0
    'Add more lines here.

Trap1:
    MsgBox "Error sending data to Excel."
    Exit Sub 'Reset error handler.
End Sub
```

See Also

DDEExecute (statement); **DDEInitiate** (function); **DDEPoke** (statement); **DDERequest**, **DDERequest\$** (functions); **DDETerminate** (statement); **DDETerminateAll** (statement); **DDETimeout** (statement).

Platform(s)

Windows.

Platform

Notes: Windows

Under Windows, the **DDEML** library is required for DDE support. This library is loaded when the first **DDEInitiate** statement is encountered and remains loaded until the **WM Basic** system is terminated. Thus, the **DDEML** library is required only if DDE statements are used within a script.

DDETerminate (statement)

Syntax `DDETerminate channel`

Description Closes the specified DDE channel.

Comments The *channel* parameter is an Integer containing the DDE channel number returned from DDEInitiate. An error will result if *channel* is invalid.

All open DDE channels are automatically terminated when the script ends.

Example 'This code fragment sets the content of the first cell in
'an Excel spreadsheet.

```
Sub Main()  
  q$ = Chr(34)  
  ch% = DDEInitiate("Excel","c:\sheets\test.xls")  
  cmd$ = "Select(" & q$ & "R1C1:R8C1" & q$ & ")"  
  DDEExecute ch%,cmd$  
  DDETerminate ch%  
End Sub
```

See Also DDEExecute (statement); DDEInitiate (function); DDEPoke (statement);
DDERequest, DDERequest\$ (functions); DDESend (function);
DDETerminateAll (statement); DDETimeout (statement).

Platform(s) Windows.

Platform Notes: Under Windows, the DDEML library is required for DDE support. This library
Windows is loaded when the first DDEInitiate statement is encountered and remains
loaded until the WM Basic system is terminated. Thus, the DDEML library is
required only if DDE statements are used within a script.

DDETerminateAll (statement)

Syntax DDETerminateAll

Description Closes all open DDE channels.

Comments All open DDE channels are automatically terminated when the script ends.

Example 'This code fragment selects the contents of the first cell in
'an Excel spreadsheet.

```
Sub Main()  
  q$ = Chr(34)  
  ch% = DDEInitiate("Excel","c:\sheets\test.xls")  
  cmd$ = "Select(" & q$ & "R1C1:R8C1" & q$ & ")"  
  DDEExecute ch%,cmd$  
  DDETerminateAll  
End Sub
```

See Also DDEExecute (statement); DDEInitiate (function); DDEPoke (statement);
DDERequest, DDERequest\$ (functions); DDESend (function); DDETerminate
(statement); DDETimeout (statement).

Platform(s) Windows.

Platform Under Windows, the DDEML library is required for DDE support. This library
Notes: is loaded when the first `DDEInitiate` statement is encountered and remains
Windows loaded until the WM Basic system is terminated. Thus, the DDEML library is
 required only if DDE statements are used within a script.

DDETimeout (statement)

Syntax `DDETimeout milliseconds`

Description Sets the number of milliseconds that must elapse before any DDE command times out.

Comments The *milliseconds* parameter is a Long and must be within the following range:

`0 <= milliseconds <= 2,147,483,647`

The default is 10,000 (10 seconds).

Example

```
Sub Main()
  q$ = Chr(34)
  ch% = DDEInitiate("Excel", "c:\sheets\test.xls")
  DDETimeout(20000)
  cmd$ = "Select(" & q$ & "R1C1:R8C1" & q$ & ")"
  DDEExecute ch%, cmd$
  DDETerminate ch%
End Sub
```

See Also `DDEExecute (statement)`; `DDEInitiate (function)`; `DDEPoke (statement)`; `DDERequest, DDERequest$ (functions)`; `DDESend (function)`; `DDETerminate (statement)`; `DDETerminateAll (statement)`.

Platform(s) Windows.

Platform Under Windows, the DDEML library is required for DDE support. This library
Notes: is loaded when the first `DDEInitiate` statement is encountered and remains
Windows loaded until the WM Basic system is terminated. Thus, the DDEML library is
 required only if DDE statements are used within a script.

Declare (statement)

Syntax `Declare {Sub | Function} name[TypeChar] [CDecl | Pascal | System | StdCall] _
 [Lib "LibName$" [Alias "AliasName$"]] [([ParameterList])] [As type]`

Where *ParameterList* is a comma-separated list of the following (up to 30 parameters are allowed):

`[Optional] [ByVal | ByRef] ParameterName[()] [As ParameterType]`

Description Creates a prototype for either an external routine or a WM Basic routine that occurs later in the source module or in another source module.

Comments `Declare` statements must appear outside of any `Sub` or `Function` declaration.

`Declare` statements are only valid during the life of the script in which they appear.

The `Declare` statement uses the following parameters:

Parameter	Description
<i>name</i>	Any valid WM Basic name. When you declare functions, you can include a type-declaration character to indicate the return type. This name is specified as a normal WM Basic keyword—i.e., it does not appear within quotes.
<i>TypeChar</i>	An optional type-declaration character used when defining the type of data returned from functions. It can be any of the following characters: #, !, \$, @, %, or &. For external functions, the @ character is not allowed. Type-declaration characters can only appear with function declarations, and take the place of the <code>As type</code> clause. Note: Currency data cannot be returned from external functions. Thus, the @ type-declaration character cannot be used when declaring external functions.
<code>CDecl</code>	Optional keyword indicating that the external subroutine or function uses the C calling convention. With C routines, arguments are pushed right to left on the stack and the caller performs stack cleanup.
<code>Pascal</code>	Optional keyword indicating that this external subroutine or function uses the Pascal calling convention. With Pascal routines, arguments are pushed left to right on the stack and the called function performs stack cleanup.
<code>System</code>	Optional keyword indicating that the external subroutine or function uses the System calling convention. With System routines, arguments are pushed right to left on the stack, the caller performs stack cleanup, and the number of arguments is specified in the <code>AL</code> register.
<code>StdCall</code>	Optional keyword indicating that the external subroutine or function uses the StdCall calling convention. With StdCall routines, arguments are pushed right to left on the stack and the called function performs stack cleanup.
<i>LibName\$</i>	Must be specified if the routine is external. This parameter specifies the name of the library or code resource containing the external routine and must appear within quotes. The <i>LibName\$</i> parameter can include an optional path specifying the exact location of the library or code resource..

AliasName\$ Alias name that must be given to provide the name of the routine if the *name* parameter is not the routine's real name. For example, the following two statements declare the same routine:

```
Declare Function GetCurrentTime Lib "user" () As Integer
Declare Function GetTime Lib "user" Alias "GetCurrentTime" _
    As Integer
```

Use an alias when the name of an external routine conflicts with the name of a WM Basic internal routine or when the external routine name contains invalid characters.

The *AliasName\$* parameter must appear within quotes.

type Indicates the return type for functions.

For external functions, the valid return types are: Integer, Long, String, Single, Double, Date, Boolean, and data objects.

Note: Currency, Variant, fixed-length strings, arrays, user-defined types, and OLE automation objects cannot be returned by external functions.

Optional Keyword indicating that the parameter is optional. All optional parameters must be of type Variant. Furthermore, all parameters that follow the first optional parameter must also be optional.

If this keyword is omitted, then the parameter being defined is required when calling this subroutine or function.

ByVal Optional keyword indicating that the caller will pass the parameter by value. Parameters passed by value cannot be changed by the called routine.

ByRef Optional keyword indicating that the caller will pass the parameter by reference. Parameters passed by reference can be changed by the called routine. If neither ByVal or ByRef are specified, then ByRef is assumed.

ParameterName Name of the parameter, which must follow WM Basic naming conventions:

1. Must start with a letter.
2. May contain letters, digits, and the underscore character (_). Punctuation and type-declaration characters are not allowed. The exclamation point (!) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character.
3. Must not exceed 80 characters in length.

Additionally, *ParameterName* can end with an optional type-declaration character specifying the type of that parameter (i.e., any of the following characters: %, &, !, #, @).

() Indicates that the parameter is an array.

ParameterType Specifies the type of the parameter (e.g., Integer, String, Variant, and so on). The `As ParameterType` clause should only be included if *ParameterName* does not contain a type-declaration character.

In addition to the default WM Basic data types, *ParameterType* can specify any user-defined structure, data object, or OLE automation object. If the data type of the parameter is not known in advance, then the `Any` keyword can be used. This forces the WM Basic compiler to relax type checking, allowing any data type to be passed in place of the given argument.

```
Declare Sub Convert Lib "mylib" (a As Any)
```

The `Any` data type can only be used when passing parameters to external routines.

Passing Parameters

By default, WM Basic passes arguments by reference. Many external routines require a value rather than a reference to a value. The `ByVal` keyword does this. For example, this C routine

```
void MessageBeep(int);
```

would be declared as follows:

```
Declare Sub MessageBeep Lib "user" (ByVal n As Integer)
```

As an example of passing parameters by reference, consider the following C routine which requires a pointer to an integer as the third parameter:

```
int SystemParametersInfo(int,int,int *,int);
```

This routine would be declared as follows (notice the `ByRef` keyword in the third parameter):

```
Declare Function SystemParametersInfo Lib "user" (ByVal action As Integer, _
    ByVal uParam As Integer,ByRef pInfo As Integer, _
    ByVal updateINI As Integer) As Integer
```

Strings can be passed by reference or by value. When they are passed by reference, a pointer to the internal handle to the WM Basic string is passed. When they are passed by value, WM Basic passes a 32-bit pointer to a null-terminated string (i.e., a C string). If an external routine modifies a passed string variable, then there must be sufficient space within the string to hold the returned characters. This can be accomplished using the `Space` function, as shown in the following example:

```
Declare Sub GetWindowsDirectory Lib "kernel" (ByVal dirname$, ByVal length%)
:
    Dim s As String
    s = Space(128)
    GetWindowsDirectory s,128
```

Another alternative to ensure that a string has sufficient space is to declare the string with a fixed length:

```
Declare Sub GetWindowsDirectory Lib "kernel" (ByVal dirname$, ByVal length%)
:
    Dim s As String * 128           'Declare a fixed-length string.
    GetWindowsDirectory s,len(s)    'Pass it to an external subroutine.
```

Calling Conventions with External Routines

For external routines, the argument list must exactly match that of the referenced routine. When calling an external subroutine or function, WM Basic needs to be told how that routine expects to receive its parameters and who is responsible for cleanup of the stack.

The following table describes which calling conventions are supported on which platform, and indicates what the default calling convention is when no explicit calling convention is specified in the `Declare` statement.

Platform	Supported Calling Conventions	Default Calling Convention
Windows	Pascal, CDecl	Pascal
Macintosh	Pascal, CDecl	Pascal

Passing Null Pointers

To pass a null pointer to an external procedure, declare the parameter that is to receive the null pointer as type Any, then pass a long value 0 by value:

```
Declare Sub Foo Lib "sample" (ByVal lpName As Any)

Sub Main()
    Sub Foo "Hello"      'Pass a 32-bit pointer to a null-terminated
string
    Sub Foo ByVal 0&     'Pass a null pointer
End Sub
```

Passing Data to External Routines

The following table shows how the different data types are passed to external routines:

Data Type	Is Passed As
ByRef Boolean	A 32-bit pointer to a 2-byte value containing -1 or 0.
ByVal Boolean	A 2-byte value containing -1 or 0.
ByVal Integer	A 32-bit pointer to a 2-byte short integer.
ByRef Integer	A 2-byte short integer.
ByVal Long	A 32-bit pointer to a 4-byte long integer.
ByRef Long	A 4-byte long integer.
ByRef Single	A 32-bit pointer to a 4-byte IEEE floating-point value (a float).
ByVal Single	A 4-byte IEEE floating-point value (a float).
ByRef Double	A 32-bit pointer to an 8-byte IEEE floating-point value (a double).
ByVal Double	An 8-byte IEEE floating-point value (a double).
ByVal String	A 32-bit pointer to a null-terminated string. With strings containing embedded nulls (Chr\$(0)), it is not possible to determine which null represents the end of the string. Therefore, the first null is considered the string terminator. An external routine can freely change the content of a string. It cannot, however, write beyond the end of the null terminator.
ByRef String	A 32-bit pointer to a 2-byte internal value representing the string. This value can only be used by external routines written specifically for WM Basic.
ByRef Date	A 32-bit pointer to an 8-byte IEEE floating-point value (a double).
ByVal Date	An 8-byte IEEE floating-point value (a double).
ByRef Currency	A 32-bit pointer to an 8-byte integer scaled by 10000.
ByVal Currency	An 8-byte integer scaled by 10000.

ByRef Variant	A 32-bit pointer to a 16-byte internal variant structure. This structure contains a 2-byte type (the same as that returned by the <code>VarType</code> function), followed by 6 bytes of slop (for alignment), followed by 8 bytes containing the value.
ByVal Variant	A 16-byte variant structure. This structure contains a 2-byte type (the same as that returned by the <code>VarType</code> function), followed by 6 bytes of slop (for alignment), followed by 8 bytes containing the value.
ByVal Object	For data objects, a 32-bit pointer to a 4-byte unsigned long integer. This value can only be used by external routines written specifically for WM Basic. For OLE automation objects, a 32-bit pointer to an <code>LPDISPATCH</code> handle is passed.
ByRef Object	For data objects, a 32-bit pointer to a 4-byte unsigned long integer that references the object. This value can only be used by external routines written specifically for BasicScript. For OLE automation objects, a 32-bit pointer to a 4-byte internal ID is passed. This value can only be used by external routines written specifically for WM Basic.
User-defined type	A 32-bit pointer to the structure. User-defined types can only be passed by reference. It is important to remember that structures in WM Basic are packed on 2-byte boundaries, meaning that the individual structure members may not be aligned consistently with similar structures declared in C.
Arrays	A 32-bit pointer to a packed array of elements of the given type. Arrays can only be passed by reference.
Dialogs	Dialogs cannot be passed to external routines. Only variable-length strings can be passed to external routines; fixed-length strings are automatically converted to variable-length strings.

WM Basic passes data to external functions consistent with that routine's prototype as defined by the `Declare` statement. There is one exception to this rule: you can override `ByRef` parameters using the `ByVal` keyword when passing individual parameters. The following example shows a number of different ways to pass an `Integer` to an external routine called `Foo`:

```
Declare Sub Foo Lib "MyLib" (ByRef i As Integer)

Sub Main
  Dim i As Integer
  i = 6
  Foo 6          'Passes a temporary integer (value 6) by reference
  Foo i          'Passes variable "i" by reference
  Foo (i)        'Passes a temporary integer (value 6) by reference
  Foo i + 1      'Passes temporary integer (value 7) by reference
  Foo ByVal i    'Passes i by value
End Sub
```

The above example shows that the only way to override passing a value by reference is to use the `ByVal` keyword.

Note: Use caution when using the `ByVal` keyword in this way. The external routine `Foo` expects to receive a pointer to an `Integer`—a 32-bit value; using `ByVal` causes WM Basic to pass the `Integer` by value—a 16-bit value. Passing data of the wrong size to any external routine will have unpredictable results.

Example `Declare` Function `IsLoaded%` Lib "Kernel" Alias "GetModuleHandle" (ByVal `name$`)

```
Declare Function GetProfileString Lib "Kernel" (ByVal SName$,ByVal
KName$, _
  ByVal Def$,ByVal Ret$,ByVal Size%) As Integer
```

```
Sub Main()
  SName$ = "Int1"          'Win.ini section name.
  KName$ = "sCountry"      'Win.ini country setting.
  ret$ = String$(255, 0)   'Initialize return string.

  If GetProfileString(SName$,KName$,"",ret$,Len(ret$)) Then
    MsgBox "Your country setting is: " & ret$
  Else
    MsgBox "There is no country setting in your win.ini file."
  End If

  If IsLoaded("Progman") Then
    MsgBox "Progman is loaded."
  Else
    MsgBox "Progman is not loaded."
  End If
End Sub
```

See Also `Call (statement)`, `Sub...End Sub (statement)`, `Function...End Function (statement)`.

Platform(s)	<p>All platforms support <code>Declare</code> for forward referencing.</p> <p>You can also use of <code>Declare</code> for referencing external routines. See below for details.</p>
Platform Notes: Windows	<p>Under Windows, external routines are contained in DLLs. The libraries containing the routines are loaded when the routine is called for the first time (i.e., not when the script is loaded). This allows a script to reference external DLLs that potentially do not exist.</p> <p>All the Windows API routines are contained in DLLs, such as "user", "kernel", and "gdi". The file extension ".exe" is implied if another extension is not given.</p> <p>If the <i>libname\$</i> parameter does not contain an explicit path to the DLL, the following search will be performed for the DLL (in this order):</p> <ol style="list-style-type: none"> 1. The current directory 2. The Windows directory 3. The Windows system directory 4. The directory containing WM Basic 5. All directories listed in the path environment variable <p>If the first character of <i>aliasname\$</i> is #, then the remainder of the characters specify the ordinal number of the routine to be called. For example, the following two statements are equivalent (under Windows, <code>GetCurrentTime</code> is defined as ordinal 15 in the user.exe module):</p> <pre> Declare Function GetTime Lib "user" Alias "GetCurrentTime" () As Integer Declare Function GetTime Lib "user" Alias "#15" () As Integer </pre> <p>Under Windows, the names of external routines declared using the <code>CDecl</code> keyword are usually preceded with an underscore character. When WM Basic searches for your external routine by name, it first attempts to load the routine exactly as specified. If unsuccessful, WM Basic makes a second attempt by prepending an underscore character to the specified name. If both attempts fail, then WM Basic generates a runtime error.</p> <p>Windows has a limitation that prevents <code>Double</code>, <code>Single</code>, and <code>Date</code> values from being returned from routines declared with the <code>CDecl</code> keyword. Routines that return data of these types should be declared <code>Pascal</code>.</p>

Platform Notes: On the Macintosh, external routines are implemented in code resources. If a code resource does not contain an explicit folder name, then WM Basic looks in the following areas:

Macintosh

1. The current folder
2. The folder containing the application
3. The Extension folder within the System folder

When using the C calling convention (with the `CDecl` keyword), WM Basic assumes 4-byte integers (the `int` data type in C). This may be problematic, as some compilers on the Macintosh assume 2-byte integers.

On the Macintosh, the code resource type is specified in *aliasname*\$ as follows:

```
" [ResourceType] $ [ResourceName] "
```

Parameter	Description
<i>ResourceType</i>	Any valid four-character name containing the type of the resource. If this parameter is omitted, then <code>CODE</code> is assumed.
<i>ResourceName</i>	<p>Name of the procedure in the code resource. If this parameter is omitted, then <i>ResourceName</i> is assumed to be the same as <i>name</i>.</p> <p>On the Macintosh, the format for parameters passed to external code resources is different than on other platforms. The differences only occur when using the Pascal calling convention (i.e., when the <code>CDecl</code> keyword is omitted). The following list describes these differences:</p> <ul style="list-style-type: none">▪ Singles, doubles, and dates passed by value or by reference are passed as a 32-bit pointer to a 10-byte value—an extended type. When passed by value, modification of the extended type does not change the original value in WM Basic.▪ Variants passed by value are passed as a 32-bit pointer to the internal variant structure used by WM Basic. Modifications to this internal structure do not affect the original value of the variable in WM Basic.▪ Currencies passed by value are passed as a 32-bit pointer to an 8-byte integer scaled by 10000.

DefType (statement)

Syntax DefInt *letterrange*
 DefLng *letterrange*
 DefStr *letterrange*
 DefSng *letterrange*
 DefDbl *letterrange*
 DefCur *letterrange*
 DefObj *letterrange*
 DefVar *letterrange*
 DefBool *letterrange*
 DefDate *letterrange*

Description Establishes the default type assigned to undeclared or untyped variables.

Comments The *DefType* statement controls automatic type declaration of variables. Normally, if a variable is encountered that hasn't yet been declared with the *Dim*, *Public*, or *Private* statement or does not appear with an explicit type-declaration character, then that variable is declared implicitly as a variant (*DefVar* A–Z). This can be changed using the *DefType* statement to specify starting letter ranges for *type* other than integer. The *letterrange* parameter is used to specify starting letters. Thus, any variable that begins with a specified character will be declared using the specified *Type*.

The syntax for *letterrange* is:

letter [-*letter*] [, *letter* [-*letter*]]...

DefType variable types are superseded by an explicit type declaration using either a type-declaration character or the *Dim*, *Public*, or *Private* statement.

The *DefType* statement only affects how WM Basic compiles scripts and has no effect at runtime.

The *DefType* statement can only appear outside all *Sub* and *Function* declarations.

The following table describes the data types referenced by the different variations of the *DefType* statement:

Statement	Data Type
DefInt	Integer
DefLng	Long
DefStr	String
DefSng	Single
DefDb1	Double
DefCur	Currency
DefObj	Object
DefVar	Variant
DefBool	Boolean
DefDate	Date

Example

```
DefStr a-l
DefLng m-r
DefSng s-u
DefDb1 v-w
DefInt x-z

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a = 100.52
    m = 100.52
    s = 100.52
    v = 100.52
    x = 100.52
    msg = "The values are:"
    msg = msg & "(String) a: " & a
    msg = msg & "(Long) m: " & m
    msg = msg & "(Single) s: " & s
    msg = msg & "(Double) v: " & v
    msg = msg & "(Integer) x: " & x
    MsgBox msg
End Sub
```

See Also Currency (data type); Date (data type); Double (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); Integer (data type).

Platform(s) Windows and Macintosh.

Desktop.ArrangeIcons (method)

Syntax Desktop.ArrangeIcons

Description Reorganizes the minimized applications on the desktop.

Example

```
Sub Main()  
    Desktop.ArrangeIcons  
End Sub
```

See Also Desktop.Cascade (method); Desktop.Tile (method).

Platform(s) Windows.

Desktop.Cascade (method)

Syntax Desktop.Cascade

Description Cascades all nonminimized windows.

Example 'This example cascades all the windows on the desktop. It first
'restores any minimized applications so that they are included in the
'cascade.

```
Sub Main()  
    Dim apps$()  
    AppList apps$  
    For i = LBound(apps) To UBound(apps)  
        AppRestore apps(i)  
    Next i  
    Desktop.Cascade  
End Sub
```

See Also Desktop.Tile (method); Desktop.ArrangeIcons (method).

Platform(s) Windows.

Desktop.SetColors (method)

Syntax Desktop.SetColors *ControlPanelItemName\$*

Description Changes the system colors to one of a predefined color set.

Example 'This example allows the user to select any of the available Windows
'color schemes.

```
Sub Main()  
    'Get color schemes from Windows  
    Dim names$()  
    ReadINISection "color schemes",names$,"CONTROL.INI"  
  
    SelectAgain: 'Allow user to select color scheme  
        item = SelectBox("Set Colors","Available Color Sets:",names$)  
        If item <> -1 Then  
            Desktop.SetColors names$(item)  
            Goto SelectAgain  
        End If  
    End Sub
```

See Also Desktop.SetWallpaper (method).

Platform(s) Windows.

Platform Under Windows, the names of the color sets are contained in the control.ini file.

Notes:

Windows

Desktop.SetWallpaper (method)

Syntax Desktop.SetWallpaper *filename\$, isTile*

Description Changes the desktop wallpaper to the bitmap specified by *filename\$*.

Comments The wallpaper will be tiled if *isTile* is True; otherwise, the bitmap will be centered on the desktop.

To remove the wallpaper, set the *filename\$* parameter to " ", as in the following example:

```
Desktop.SetWallpaper " ", True
```


Example 'This example reads a list of .BMP files from the Windows directory
'and allows the user to select any of these as wallpaper.

```
Sub Main()
    Dim list$()

    ' Create the prefix for the bitmap filenames
    d$ = System.WindowsDirectory$
    If Right(d$,1) <> "\" Then d$ = d$ & "\"
    f$ = d$ & "*.BMP"

    FileList list$,f$ 'Get list of bitmaps from Windows directory

    ' Were there any bitmaps?
    If ArrayDims(list$) = 0 Then
        MsgBox "There aren't any bitmaps in the Windows directory"
        Exit Sub
    End If

    'Add "(none)"
    ReDim Preserve list$ (UBound(list$) + 1)
    list$(UBound(list$)) = "(none)"

SelectAgain:    'Allow user to select item
    item = SelectBox("Set Wallpaper","Available Wallpaper:",list$)

    Select Case item
        Case -1
            End
        Case UBound(list$)
            Desktop.SetWallPaper "",True
            Goto SelectAgain
        Case Else
            Desktop.SetWallPaper d$ & list$(item),True
            Goto SelectAgain
    End Select
End Sub
```

See Also Desktop.SetColors (method).

Platform(s) Windows.

Platform Under Windows, the Desktop.SetWallpaper method makes permanent
Notes: changes to the wallpaper by writing the new wallpaper information to the
Windows win.ini file.

Desktop.Snapshot (method)

Syntax Desktop.Snapshot [*spec*]

Description Takes a snapshot of a particular section of the screen and saves it to the Clipboard.

Comments The *spec* parameter is an Integer specifying the screen area to be saved. It can be any of the following:

- 0 Entire screen
- 1 Client area of the active application
- 2 Entire window of the active application
- 3 Client area of the active window
- 4 Entire window of the active window

Before the snapshot is taken, each application is updated. This ensures that any application that is in the middle of drawing will have a chance to finish before the snapshot is taken.

There is a slight delay if the specified window is large.

Example 'This example takes a snapshot of Program Manager and pastes the resulting bitmap into Windows Paintbrush.

```
Sub Main()  
    AppActivate "Program Manager"      'Activate Program Manager.  
    Desktop.Snapshot 2                 'Place snapshot into Clipboard.  
    id = Shell("pbrush")               'Run Paintbrush.  
    Menu "Edit.Paste"                 'Paste snapshot into Paintbrush.  
End Sub
```

Platform(s) Windows.

Platform Under Windows, pictures are placed into the Clipboard in bitmap format.

Notes:
Windows

Desktop.Tile (method)

Syntax Desktop.Tile

Description Tiles all nonminimized windows.

Example 'This example tiles all the windows on the desktop. It first restores any minimized applications so that they are included in the tile.

```
Sub Main()  
    Dim apps$()  
    AppList apps$  
    For i = LBound(apps) To UBound(apps)  
        AppRestore apps(i)  
    Next i  
    Desktop.Tile  
End Sub
```

See Also Desktop.Cascade (method); Desktop.ArrangeIcons (method).

Platform(s) Windows.

Dialog (function)

Syntax `Dialog(DialogVariable [, [DefaultButton] [, Timeout]])`

Description Displays the dialog box associated with *DialogVariable*, returning an Integer indicating which button was clicked.

Comments The Dialog function returns any of the following values:

- 1 The OK button was clicked.
- 0 The Cancel button was clicked.
- >0 A push button was clicked. The returned number represents which button was clicked based on its order in the dialog box template (1 is the first push button, 2 is the second push button, and so on).

The Dialog function accepts the following parameters:

Parameter	Description
-----------	-------------

<i>DialogVariable</i>	Name of a variable that has previously been dimensioned as a user dialog box. This is accomplished using the Dim statement:
-----------------------	---

```
Dim MyDialog As MyTemplate
```

All dialog variables are local to the Sub or Function in which they are defined. Private and public dialog variables are not allowed.

DefaultButton An Integer specifying which button is to act as the default button in the dialog box. The value of *DefaultButton* can be any of the following:

_2	This value indicates that there is no default button.
_1	This value indicates that the OK button, if present, should be used as the default.
0	This value indicates that the Cancel button, if present, should be used as the default.
>0	This value indicates that the <i>N</i> th button should be used as the default. This number is the index of a push button within the dialog box template.

If *DefaultButton* is not specified, then *_1* is used. If the number specified by *DefaultButton* does not correspond to an existing button, then there will be no default button.

The default button appears with a thick border and is selected when the user presses Enter on a control other than a push button.

Timeout An Integer specifying the number of milliseconds to display the dialog box before automatically dismissing it. If *TimeOut* is not specified or is equal to 0, then the dialog box will be displayed until dismissed by the user.

If a dialog box has been dismissed due to a timeout, the *Dialog* function returns 0.

Example 'This example displays an abort/retry/ignore disk error dialog box.

```
Sub Main()  
    Begin Dialog DiskErrorTemplate 16,32,152,48,"Disk Error"  
        Text 8,8,100,8,"The disk drive door is open."  
        PushButton 8,24,40,14,"Abort",.Abort  
        PushButton 56,24,40,14,"Retry",.Retry  
        PushButton 104,24,40,14,"Ignore",.Ignore  
    End Dialog  
    Dim DiskError As DiskErrorTemplate  
    r% = Dialog(DiskError,3,0)  
    MsgBox "You selected button: " & r%  
End Sub
```

See Also CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (statement); DropDownListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Platform(s) Windows and Macintosh.

Dialog (statement)

Syntax `Dialog DialogVariable [, [DefaultButton] [, Timeout]]`

Description Same as the Dialog function, except that the Dialog statement does not return a value. (See Dialog [function].)

Example 'This example displays an abort/retry/ignore disk error dialog box.

```
Sub Main()
    Begin Dialog DiskErrorTemplate 16,32,152,48,"Disk Error"
        Text 8,8,100,8,"The disk drive door is open."
        PushButton 8,24,40,14,"Abort",.Abort
        PushButton 56,24,40,14,"Retry",.Retry
        PushButton 104,24,40,14,"Ignore",.Ignore
    End Dialog
    Dim DiskError As DiskErrorTemplate
    Dialog DiskError,3,0
End Sub
```

See Also Dialog (function).

Platform(s) Windows and Macintosh.

Dim (statement)

Syntax `Dim name [(<subscripts>)] [As [New] type] [, name [(<subscripts>)] [As [New] type]]...`

Description Declares a list of local variables and their corresponding types and sizes.

Comments If a type-declaration character is used when specifying *name* (such as %, @, &, \$, or !), the optional [As *type*] expression is not allowed. For example, the following are allowed:

```
Dim Temperature As Integer
Dim Temperature%
```

The *subscripts* parameter allows the declaration of dynamic and fixed arrays.

The *subscripts* parameter uses the following syntax:

```
[lower to] upper [, [lower to] upper]...
```

The *lower* and *upper* parameters are integers specifying the lower and upper bounds of the array. If *lower* is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). WM Basic supports a maximum of 60 array dimensions.

The total size of an array (not counting space for strings) is limited to 64K.

Dynamic arrays are declared by not specifying any bounds:

```
Dim a()
```

The *type* parameter specifies the type of the data item being declared. It can be any of the following data types: `String`, `Integer`, `Long`, `Single`, `Double`, `Currency`, `Object`, data object, built-in data type, or any user-defined data type.

A `Dim` statement within a subroutine or function declares variables local to that subroutine or function. If the `Dim` statement appears outside of any subroutine or function declaration, then that variable has the same scope as variables declared with the `Private` statement.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the `String` type-declaration character:

```
Dim name As String * length
```

where *length* is a literal number specifying the string's length.

Implicit Variable Declaration

If WM Basic encounters a variable that has not been explicitly declared with `Dim`, then the variable will be implicitly declared using the specified type-declaration character (`#`, `%`, `@`, `$`, or `&`). If the variable appears without a type-declaration character, then the first letter is matched against any pending `DefType` statements, using the specified type if found. If no `DefType` statement has been encountered corresponding to the first letter of the variable name, then `Variant` is used.

Creating New Objects

The optional `New` keyword is used to declare a new instance of the specified data object. This keyword can only be used with data object types. Furthermore, this keyword cannot be used when declaring arrays.

At runtime, the application or extension that defines that object type is notified that a new object is being defined. The application responds by creating a new physical object (within the appropriate context) and returning a reference to that object, which is immediately assigned to the variable being declared.

When that variable goes out of scope (i.e., the `Sub` or `Function` procedure in which the variable is declared ends), the application is notified. The application then performs some appropriate action, such as destroying the physical object.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value
Integer	0
Long	0
Double	0.0
Single	0.0
Date	December 31, 1899 00:00:00
Currency	0.0
Boolean	False
Object	Nothing
Variant	Empty
String	" " (zero-length string)

User-defined type Each element of the structure is given an initial value, as described above.

Arrays Each element of the array is given an initial value, as described above.

Naming Conventions

Variable names must follow these naming rules:

1. Must start with a letter.
2. May contain letters, digits, and the underscore character (_); punctuation is not allowed. The exclamation point (!) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character.

The last character of the name can be any of the following type-declaration characters: #, @, %, !, &, and \$.

3. Must not exceed 80 characters in length.
4. Cannot be a reserved word.

Examples 'The following examples use the Dim statement to declare various 'variable types.

```
Sub Main()  
    Dim i As Integer  
    Dim l& 'long  
    Dim s As Single  
    Dim d# 'double  
    Dim c$ 'string  
    Dim MyArray(10) As Integer '10 element integer array  
    Dim MyStrings$(2,10) '2-10 element string arrays  
    Dim Filenames$(5 to 10) '6 element string array  
    Dim Values(1 to 10, 100 to 200) '111 element variant array  
End Sub
```

See Also Redim (statement); Public (statement); Private (statement); Option Base (statement).

Platform(s) Windows and Macintosh.

Dir, Dir\$ (functions)

Syntax Dir\$[(*filespec\$* [, *attributes*])]

Description Returns a String containing the first or next file matching *filespec\$*.

If *filespec\$* is specified, then the first file matching that *filespec\$* is returned. If *filespec\$* is not specified, then the next file matching the initial *filespec\$* is returned.

Comments Dir\$ returns a String, whereas Dir returns a String variant.

The Dir\$/Dir functions take the following parameters:

Parameter	Description
<i>filespec\$</i>	<p>String containing a file specification.</p> <p>If this parameter is specified, then Dir\$ returns the first file matching this file specification. If this parameter is omitted, then the next file matching the initial file specification is returned.</p> <p>If no path is specified in <i>filespec\$</i>, then the current directory is used.</p>
<i>attributes</i>	<p>Integer specifying attributes of files you want included in the list, as described below. If omitted, then only the normal, read-only, and archive files are returned.</p> <p>An error is generated if Dir\$ is called without first calling it with a valid <i>filespec\$</i>.</p> <p>If there is no matching <i>filespec\$</i>, then a zero-length string is returned.</p>

Wildcards

The *filespec\$* argument can include wildcards, such as * and ?. The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complete searching patterns. The following table shows some examples:

This pattern	Matches these files	Doesn't match these files
S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT
C*T.TXT	CAT.TXT	CAP.TXT ACATS.TXT
C*T	CAT	CAT.DOC CAP.TXT
C?T	CAT CUT	CAT.TXT CAPIT CT
*	(All files)	

Attributes

You can control which files are included in the search by specifying the optional *attributes* parameter. The *Dir*, *Dir\$* functions always return all normal, read-only, and archive files (*ebNormal* Or *ebReadOnly* Or *ebArchive*). To include additional files, you can specify any combination of the following attributes (combined with the *Or* operator):

Constant	Value	Includes
<i>ebNormal</i>	0	Normal, Read-only, and archive files
<i>ebHidden</i>	2	Hidden files
<i>ebSystem</i>	4	System files
<i>ebVolume</i>	8	Volume label
<i>ebDirectory</i>	16	DOS subdirectories

Example 'This example dimensions a null array and fills it with directory 'entries. The result is displayed in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$(10)
    a(1) = Dir$("*.*")
    i% = 1
    While (a(i%) <> "") And (i% < 10)
        i% = i% + 1
        a(i%) = Dir$
    Wend
    MsgBox a(1) & crlf & a(2) & crlf & a(3) & crlf & a(4)
End Sub
```

See Also ChDir (statement); ChDrive (statement); CurDir, CurDir\$ (functions); Mkdir (statement); Rmdir (statement); FileList (statement).

Platform(s) Windows and Macintosh.

Platform The Macintosh does not support wildcard characters such as * and ?. These are
Notes: valid filename characters. Instead of wildcards, the Macintosh uses the MacID
Macintosh function to specify a collection of files of the same type. The syntax for this function is:

```
Dir$(filespec$, MacID(text$))
```

The *text\$* parameter is a four-character string containing a file type, a resource type, an application signature, or an Apple event. A runtime error occurs if the MacID function is used on platforms other than the Macintosh.

When the MacID function is used, the *filespec\$* parameter specifies the directory in which to search for files of the indicated type.

Platform Notice that WM Basic's filename matching is different than DOS's. The pattern
Notes: "*.*" under DOS matches all files. With WM Basic, this pattern matches only
Windows files that have file extensions.

DiskDrives (statement)

Syntax DiskDrives array()

Description Fills the specified String or Variant array with a list of valid drive letters.

Comments The `array()` parameter specifies either a zero- or a one-dimensional array of strings or variants. The array can be either dynamic or fixed.

If `array()` is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the `LBound`, `UBound`, and `ArrayDims` functions to determine the number and size of the new array's dimensions.

If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for `String` arrays) or `Empty` (for `Variant` arrays). A runtime error results if the array is too small to hold the new elements.

Example 'This example builds and displays an array containing the first three 'available disk drives.

```
Sub Main()
    Dim drive$()
    DiskDrives drive$
    r% = SelectBox("Available Disk Drives",,drive$)
End Sub
```

See Also `ChDrive` (statement); `DiskFree` (function).

Platform(s) Windows.

DiskFree (function)

Syntax `DiskFree&([drive$])`

Description Returns a `Long` containing the free space (in bytes) available on the specified drive.

Comments If `drive$` is zero-length or not specified, then the current drive is assumed.

Only the first character of the `drive$` string is used.

Example 'This example uses `DiskFree` to set the value of `i` and then displays the 'result in a message box.

```
Sub Main()
    s$ = "c"
    i# = DiskFree(s$)
    MsgBox "Free disk space on drive '" & s$ & "' is: " & i#
End Sub
```

See Also `ChDrive` (statement); `DiskDrives` (statement).

Platform(s) Windows and Macintosh.

Platform On systems that do not support drive letters, the *drive\$* parameter specifies the
Notes: name of the path from which to retrieve the free disk space.

DlgControlId (function)

Syntax DlgControlId(*ControlName\$*)

Description Returns an Integer containing the index of the specified control as it appears in the dialog box template.

Comments The first control in the dialog box template is at index 0, the second is at index 1, and so on.

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with that control in the dialog box template.

The WM Basic statements and functions that dynamically manipulate dialog box controls identify individual controls using either the *.Identifier* name of the control or the control's index. Using the index to refer to a control is slightly faster but results in code that is more difficult to maintain.

Example

```
Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    'If a control is clicked, disable the next three controls.
    If Action% = 2 Then
        'Enable the next three controls.
        start% = DlgControlId(ControlName$)

        For i = start% + 1 To start% + 3
            DlgEnable i,True
        Next i
        DlgProc = 1      'Don't close the dialog box.
    End If
End Function
```

See Also DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgEnable (function)

Syntax DlgEnable(*ControlName\$* | *ControlIndex*)

Description Returns True if the specified control is enabled; returns False otherwise.

Comments Disabled controls are dimmed and cannot receive keyboard or mouse input.

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

You cannot disable the control with the focus.

Example

```
If DlgEnable("SaveOptions") Then
    MsgBox "The Save Options are enabled."
End If
If DlgEnable(10) And DlgVisible(12) Then code = 1 Else code = 2
```

See Also DlgControl (statement); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgEnable (statement)

Syntax DlgEnable {*ControlName\$* | *ControlIndex*} [, *isOn*]

Description Enables or disables the specified control.

Comments Disabled controls are dimmed and cannot receive keyboard or mouse input.

The *isOn* parameter is an Integer specifying the new state of the control. It can be any of the following values:

0 The control is disabled.

1 The control is enabled.

Omitted Toggles the control between enabled and disabled.

Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example

```

DlgEnable "SaveOptions", False    'Disable the Save Options control.
DlgEnable "EditingOptions"        'Toggle a group of option buttons.

For i = 0 To 5
    DlgEnable i,True              'Enable six controls.
Next i

```

See Also DlgControl (statement); DlgEnable (function); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgFocus (function)

Syntax DlgFocus\$()

Description Returns a String containing the name of the control with the focus.

Comments The name of the control is the *.Identifier* parameter associated with the control in the dialog box template.

Example

```

'This code fragment makes sure that the control being disabled does not
'currently have the focus (otherwise, a runtime error would occur).

If DlgFocus$ = "Files" Then    'Does it have the focus?
    DlgFocus "OK"              'Change the focus to another control.
End If
DlgEnable "Files", False      'Now we can disable the control.

```

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgFocus (statement)

Syntax DlgFocus ControlName\$ | ControlIndex

Description Sets focus to the specified control.

Comments A runtime error results if the specified control is hidden, disabled, or nonexistent.

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example 'This code fragment makes sure that the control being disabled does
'not currently have the focus (otherwise, a runtime error would occur).

If DlgFocus\$ = "Files" Then 'Does it have the focus?
 DlgFocus "OK" 'Change the focus to another control.
End If
DlgEnable "Files", False 'Now we can disable the control.

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement);
DlgFocus (function); DlgListBoxArray (function); DlgListBoxArray
(statement); DlgSetPicture (statement); DlgText (statement); DlgText
(function); DlgValue (function); DlgValue (statement); DlgVisible
(statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgListBoxArray (function)

Syntax DlgListBoxArray({ *ControlName\$* | *ControlIndex*}, *ArrayVariable*)

Description Fills a list box, combo box, or drop list box with the elements of an array, returning an Integer containing the number of elements that were actually set into the control.

Comments The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

The *ArrayVariable* parameter specifies a single-dimensioned array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. *ArrayVariable* can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.

Example 'This dialog function refills an array with files.

```
Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 And ControlName$ = "Files" Then
        Dim NewFiles$()           'Create a new dynamic array.
        FileList NewFiles$,"*.txt" 'Fill the array with files.
        r% = DlgListBoxArray "Files",NewFiles$ 'Set items in the list
box.
        DlgValue "Files",0        'Set the selection to the first item.
        DlgProc = 1              'Don't close the dialog box.
    End If
    MsgBox r% & " items were added to the list box."
End Function
```

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement);
DlgFocus (function); DlgFocus (statement); DlgListBoxArray (statement);
DlgSetPicture (statement); DlgText (statement); DlgText (function);
DlgValue (function); DlgValue (statement); DlgVisible (statement);
DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgListBoxArray (statement)

Syntax DlgListBoxArray {ControlName\$ | ControlIndex}, ArrayVariable

Description Fills a list box, combo box, or drop list box with the elements of an array.

Comments The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

The *ArrayVariable* parameter specifies a single-dimensioned array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. *ArrayVariable* can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.

Example 'This dialog function refills an array with files.

```
Function DlgProc(ControlName$,Action%,SuppValue%) As Integer
    If Action% = 2 And ControlName$ = "Files" Then
        Dim NewFiles$()           'Create a new dynamic
array.
        FileList NewFiles$,"*.txt" 'Fill the array with files.
        DlgListBoxArray "Files",NewFiles$ 'Set items in the list box.
        DlgValue "Files",0         'Set the selection to the
first item.
    End If
End Function
```

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgProc (function)

Syntax Function *DlgProc*(*ControlName*\$, *Action*, *SuppValue*) As Integer

Description Describes the syntax, parameters, and return value for dialog functions.

Comments Dialog functions are called by WM Basic during the processing of a custom dialog box. The name of a dialog function (*DlgProc*) appears in the Begin Dialog statement as the *.DlgProc* parameter.

Dialog functions require the following parameters:

Parameter	Description
<i>ControlName</i> \$	String containing the name of the control associated with <i>Action</i> .
<i>Action</i>	Integer containing the action that called the dialog function.
<i>SuppValue</i>	Integer of extra information associated with <i>Action</i> . For some actions, this parameter is not used.
	When WM Basic displays a custom dialog box, the user may click on buttons, type text into edit fields, select items from lists, and perform other actions. When these actions occur, WM Basic calls the dialog function, passing it the action, the name of the control on which the action occurred, and any other relevant information associated with the action.

The following table describes the different actions sent to dialog functions:

Action	Description
1	<p>This action is sent immediately before the dialog box is shown for the first time. This gives the dialog function a chance to prepare the dialog box for use. When this action is sent, <i>ControlName\$</i> contains a zero-length string, and <i>SuppValue</i> is 0.</p> <p>The return value from the dialog function is ignored in this case.</p> <p>Before Showing the Dialog Box</p> <p>After action 1 is sent, WM Basic performs additional processing before the dialog box is shown. Specifically, it cycles through the dialog box controls checking for visible picture or picture button controls. For each visible picture or picture button control, WM Basic attempts to load the associated picture.</p> <p>In addition to checking picture or picture button controls, WM Basic will automatically hide any control outside the confines of the visible portion of the dialog box. This prevents the user from tabbing to controls that cannot be seen. However, it does not prevent you from showing these controls with the <code>DlgVisible</code> statement in the dialog function.</p>
2	<p>This action is sent when:</p> <ul style="list-style-type: none">▪ A button is clicked, such as OK, Cancel, or a push button. In this case, <i>ControlName\$</i> contains the name of the button. <i>SuppValue</i> contains 1 if an OK button was clicked and 2 if a Cancel button was clicked; <i>SuppValue</i> is undefined otherwise. <p>If the dialog function returns 0 in response to this action, then the dialog box will be closed. Any other value causes WM Basic to continue dialog processing.</p> <ul style="list-style-type: none">▪ A check box's state has been modified. In this case, <i>ControlName\$</i> contains the name of the check box, and <i>SuppValue</i> contains the new state of the check box (1 if on, 0 if off).▪ An option button is selected. In this case, <i>ControlName\$</i> contains the name of the option button that was clicked, and <i>SuppValue</i> contains the index of the option button within the option button group (0-based).▪ The current selection is changed in a list box, drop list box, or combo box. In this case, <i>ControlName\$</i> contains the name of the list box, combo box, or drop list box, and <i>SuppValue</i> contains the index of the new item (0 is the first item, 1 is the second, and so on).

- 3 This action is sent when the content of a text box or combo box has been changed. This action is only sent when the control loses focus. When this action is sent, *ControlName\$* contains the name of the text box or combo box, and *SuppValue* contains the length of the new content.
The dialog function's return value is ignored with this action.
- 4 This action is sent when a control gains the focus. When this action is sent, *ControlName\$* contains the name of the control gaining the focus, and *SuppValue* contains the index of the control that lost the focus (0-based).
The dialog function's return value is ignored with this action.
- 5 This action is sent continuously when the dialog box is idle. If the dialog function returns 1 in response to this action, then the idle action will continue to be sent. If the dialog function returns 0, then WM Basic will not send any additional idle actions.
When the idle action is sent, *ControlName\$* contains a zero-length string, and *SuppValue* contains the number of times the idle action has been sent so far.
- 6 This action is sent when the dialog box is moved. The *ControlName\$* parameter contains a zero-length string, and *SuppValue* is 0.
The dialog function's return value is ignored with this action.
User-defined dialog boxes cannot be nested. In other words, the dialog function of one dialog box cannot create another user-defined dialog box. You can, however, invoke any built-in dialog box, such as `MsgBox` or `InputBox$`.
Within dialog functions, you can use the following additional WM Basic statements and functions. These statements allow you to manipulate the dialog box controls dynamically.

<code>DlgVisible</code>	<code>DlgText\$</code>	<code>DlgText</code>	
<code>DlgSetPicture</code>		<code>DlgListBoxArray</code>	<code>DlgFocus</code>
<code>DlgEnable</code>	<code>DlgControlId</code>		

For compatibility with previous versions of WM Basic, the dialog function can optionally be declared to return a `Variant`. When returning a variable, WM Basic will attempt to convert the variant to an `Integer`. If the returned variant cannot be converted to an `Integer`, then 0 is assumed to be returned from the dialog function.

Example 'This dialog function enables/disables a group of option buttons
'when a check box is clicked.

```
Function SampleDlgProc(ControlName$, Action%, SuppValue%)
    If Action% = 2 And ControlName$ = "Printing" Then
        DlgEnable "PrintOptions",SuppValue%
        SampleDlgProc = 1    'Don't close the dialog box.
    End If
End Function

Sub Main()
    Begin Dialog SampleDialogTemplate
34,39,106,45,"Sample",.SampleDlgProc
        OKButton 4,4,40,14
        CancelButton 4,24,40,14
        CheckBox 56,8,38,8,"Printing",.Printing
        OptionGroup .PrintOptions
            OptionButton 56,20,51,8,"Landscape",.Landscape
            OptionButton 56,32,40,8,"Portrait",.Portrait
        End Dialog
    Dim SampleDialog As SampleDialogTemplate
    SampleDialog.Printing = 1
    r% = Dialog(SampleDialog)
End Sub
```

See Also Begin Dialog (statement).

Platform(s) Windows and Macintosh.

DlgSetPicture (statement)

Syntax DlgSetPicture {ControlName\$ | ControlIndex},PictureName\$,PictureType

Description Changes the content of the specified picture or picture button control.

Comments The DlgSetPicture statement accepts the following parameters:

Parameter	Description				
<i>ControlName\$</i>	String containing the name of the <i>.Identifier</i> parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specified control within the template. Alternatively, by specifying the <i>ControlIndex</i> parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).				
<i>PictureName\$</i>	String containing the name of the picture. If <i>PictureType</i> is 0, then this parameter specifies the name of the file containing the image. If <i>PictureType</i> is 10, then <i>PictureName\$</i> specifies the name of the image within the resource of the picture library. If <i>PictureName\$</i> is empty, then the current picture associated with the specified control will be deleted. Thus, a technique for conserving memory and resources would involve setting the picture to empty before hiding a picture control.				
<i>PictureType</i>	Integer specifying the source for the image. The following sources are supported: <table> <tr> <td>0</td><td>The image is contained in a file on disk.</td></tr> <tr> <td>10</td><td>The image is contained in the picture library specified by the Begin Dialog statement. When this type is used, the <i>PictureName\$</i> parameter must be specified with the Begin Dialog statement.</td></tr> </table>	0	The image is contained in a file on disk.	10	The image is contained in the picture library specified by the Begin Dialog statement. When this type is used, the <i>PictureName\$</i> parameter must be specified with the Begin Dialog statement.
0	The image is contained in a file on disk.				
10	The image is contained in the picture library specified by the Begin Dialog statement. When this type is used, the <i>PictureName\$</i> parameter must be specified with the Begin Dialog statement.				

Examples

```
DlgSetPicture "Picture1", "\windows\checks.bmp", 0 'Set picture from a
file.

DlgSetPicture 27, "FaxReport", 10 'Set control 10's
image
'from a library.
```

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function), Picture (statement), PictureBox (statement).

Platform(s) Windows and Macintosh.

Platform Notes: Under Windows, picture controls can contain either bitmaps or WMFs (Windows metafiles). When extracting images from a picture library, WM Basic assumes that the resource type for metafiles is 256.
Picture libraries are implemented as DLLs on the Windows.

Platform Notes: Picture controls on the Macintosh can contain only PICT images. These are contained in files of type PICT.

Macintosh Picture libraries on the Macintosh are files with collections of named PICT resources. The *PictureName\$* parameter corresponds to the name of one the resources as it appears within the file.

DlgText (statement)

Syntax DlgText {*ControlName\$* | *ControlIndex*}, *NewText\$*

Description Changes the text content of the specified control.

Comments The effect of this statement depends on the type of the specified control:

Control Type	Effect of DlgText
Picture	Runtime error.
Option group	Runtime error.
Drop list box	Sets the current selection to the item matching <i>NewText\$</i> . If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with <i>NewText\$</i> . If no match is found, then the selection is removed.
OK button	Sets the label of the control to <i>NewText\$</i> .
Cancel button	Sets the label of the control to <i>NewText\$</i> .
Push button	Sets the label of the control to <i>NewText\$</i> .
List box	Sets the current selection to the item matching <i>NewText\$</i> . If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with <i>NewText\$</i> . If no match is found, then the selection is removed.
Combo box	Sets the content of the edit field of the combo box to <i>NewText\$</i> .
Text	Sets the label of the control to <i>NewText\$</i> .
Text box	Sets the content of the text box to <i>NewText\$</i> .
Group box	Sets the label of the control to <i>NewText\$</i> .
Option button	Sets the label of the control to <i>NewText\$</i> .

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example `DlgText "GroupBox1","Save Options" 'Change text of group box 1.`
`If DlgText$(9) = "Save Options" Then`
`DlgText 9,"Editing Options" 'Change text to "Editing`
`Options".`
`End If`

See Also `DlgControl (statement); DlgEnable (function); DlgEnable (statement);`
`DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function);`
`DlgListBoxArray (statement); DlgSetPicture (statement); DlgText`
`(function); DlgValue (function); DlgValue (statement); DlgVisible`
`(statement); DlgVisible (function).`

Platform(s) Windows and Macintosh.

DlgText\$ (function)

Syntax `DlgText$(ControlName$ | ControlIndex)`

Description Returns the text content of the specified control.

Comments The text returned depends on the type of the specified control:

Control Type	Value Returned by DlgText\$
Picture	No value is returned. A runtime error occurs.
Option group	No value is returned. A runtime error occurs.
Drop list box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
OK button	Returns the label of the control.
Cancel button	Returns the label of the control.
Push button	Returns the label of the control.
List box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
Combo box	Returns the content of the edit field portion of the combo box.
Text	Returns the label of the control.
Text box	Returns the content of the control.
Group box	Returns the label of the control.
Option button	Returns the label of the control.

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example `MsgBox DlgText$(10) 'Display the text in the tenth control.`
`If DlgText$("SaveOptions") = "EditingOptions" Then`
 `MsgBox "You are currently viewing the editing options."`
`End If`

See Also `DlgControl (statement); DlgEnable (function); DlgEnable (statement);`
`DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function);`
`DlgListBoxArray (statement); DlgSetPicture (statement); DlgText`
`(statement); DlgValue (function); DlgValue (statement); DlgVisible`
`(statement); DlgVisible (function).`

Platform(s) Windows and Macintosh.

DlgValue (function)

Syntax `DlgValue(ControlName$ | ControlIndex)`

Description Returns an Integer indicating the value of the specified control.

Comments The value of any given control depends on its type, according to the following table:

Control Type	DlgValue Returns
Option group	The index of the selected option button within the group (0 is the first option button, 1 is the second, and so on).
List box	The index of the selected item.
Drop list box	The index of the selected item.
Check box	1 if the check box is checked; 0 otherwise.
	A runtime error is generated if <code>DlgValue</code> is used with controls other than those listed in the above table.
	The <i>ControlName\$</i> parameter contains the name of the <i>.Identifier</i> parameter associated with a control in the dialog box template. Alternatively, by specifying the <i>ControlIndex</i> parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

See **Example** (statement).

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (statement); DlgVisible (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgValue (statement)

Syntax DlgValue { *ControlName\$* | *ControlIndex* }, *Value*

Description Changes the value of the given control.

Comments The value of any given control is an Integer and depends on its type, according to the following table:

Control Type	Description of Value
Option group	The index of the new selected option button within the group (0 is the first option button, 1 is the second, and so on).
List box	The index of the new selected item.
Drop list box	The index of the new selected item.
Check box	1 if the check box is to be checked; 0 if the check is to be removed.

A runtime error is generated if DlgValue is used with controls other than those listed in the above table.

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example 'This code fragment toggles the value of a check box.

```
If DlgValue("MyCheckBox") = 1 Then
    DlgValue "MyCheckBox",0
Else
    DlgValue "MyCheckBox",1
End If
```

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgVisible (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgVisible (function)

Syntax DlgVisible(*ControlName\$* | *ControlIndex*)

Description Returns True if the specified control is visible; returns False otherwise .

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the template (0 is the first control in the template, 1 is the second, and so on).

A runtime error is generated if DlgVisible is called with no user dialog is active.

Example If DlgVisible("Portrait") Then Beep
 If DlgVisible(10) And DlgVisible(12) Then
 MsgBox "The 10th and 12th controls are visible."
 End If

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (function).

Platform(s) Windows and Macintosh.

DlgVisible (statement)

Syntax DlgVisible {*ControlName\$* | *ControlIndex*} [, *isOn*]

Description Hides or shows the specified control.

Comments Hidden controls cannot be seen in the dialog box and cannot receive the focus using Tab.

The *isOn* parameter is an Integer specifying the new state of the control. It can be any of the following values:

- 1 The control is shown.
- 0 The control is hidden.
- Omitted Toggles the visibility of the control.

Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).

The *ControlName\$* parameter contains the name of the *.Identifier* parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the *ControlIndex* parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Picture Caching

When the dialog box is first created and before it is shown, WM Basic calls the dialog function with *action* set to 1. At this time, no pictures have been loaded into the picture controls contained in the dialog box template. After control returns from the dialog function and before the dialog box is shown, WM Basic will load the pictures of all visible picture controls. Thus, it is possible for the dialog function to hide certain picture controls, which prevents the associated pictures from being loaded and causes the dialog box to load faster. When a picture control is made visible for the first time, the associated picture will then be loaded.

Example 'This example creates a dialog box with two panels. The DlgVisible
'statement is used to show or hide the controls of the different
'panels.

```
Sub EnableGroup(start%, finish%)
    For i = 6 To 13                                'Disable all options.
        DlgVisible i, False
    Next i
    For i = start% To finish%                        'Enable only the right ones.
        DlgVisible i, True
    Next i
End Sub

Function DlgProc(ControlName$, Action%, SuppValue%)
    If Action% = 1 Then
        DlgValue "WhichOptions",0                'Set to save options.
        EnableGroup 6, 8                          'Enable the save options.
    End If
    If Action% = 2 And ControlName$ = "SaveOptions" Then
        EnableGroup 6, 8                          'Enable the save options.
        DlgProc = 1                               'Don't close the dialog box.
    End If
    If Action% = 2 And ControlName$ = "EditingOptions" Then
        EnableGroup 9, 13                         'Enable the editing options.
        DlgProc = 1                               'Don't close the dialog box.
    End If
End Function

Sub Main()
    Begin Dialog OptionsTemplate 33, 33, 171, 134, "Options", .DlgProc
        'Background (controls 0-5)
        GroupBox 8, 40, 152, 84, ""
        OptionGroup .WhichOptions
            OptionButton 8, 8, 59, 8, "Save Options",.SaveOptions
            OptionButton 8, 20, 65, 8, "Editing Options",.EditingOptions
        OKButton 116, 7, 44, 14
        CancelButton 116, 24, 44, 14

        'Save options (controls 6-8)
        CheckBox 20, 56, 88, 8, "Always create backup",.CheckBox1
        CheckBox 20, 68, 65, 8, "Automatic save",.CheckBox2
        CheckBox 20, 80, 70, 8, "Allow overwriting",.CheckBox3

        'Editing options (controls 9-13)
        CheckBox 20, 56, 65, 8, "Overtyping mode",.OvertypingMode
        CheckBox 20, 68, 69, 8, "Uppercase only",.UppercaseOnly
        CheckBox 20, 80, 105, 8, "Automatically check
syntax",.AutoCheckSyntax
        CheckBox 20, 92, 73, 8, "Full line selection",.FullLineSelection
        CheckBox 20, 104, 102, 8, "Typing replaces
selection",.TypingReplacesText
    End Dialog

    Dim OptionsDialog As OptionsTemplate
    Dialog OptionsDialog
End Sub
```

See Also DlgControl (statement); DlgEnable (function); DlgEnable (statement); DlgFocus (function); DlgFocus (statement); DlgListBoxArray (function); DlgListBoxArray (statement); DlgSetPicture (statement); DlgText (statement); DlgText (function); DlgValue (function); DlgValue (statement); DlgVisible (statement).

Platform(s) Windows and Macintosh.

Do...Loop (statement)

Syntax 1 Do {While | Until} *condition* statements Loop

Syntax 2 Do
 statements
Loop {While | Until} *condition*

Syntax 3 Do
 statements
Loop

Description Repeats a block of WM Basic statements while a condition is True or until a condition is True.

Comments If the {While | Until} conditional clause is not specified, then the loop repeats the statements forever (or until WM Basic encounters an Exit Do statement).

The *condition* parameter specifies any Boolean expression.

Examples

```
Sub Main()
    'This first example uses the Do...While statement, which performs
    'the iteration, then checks the condition, and repeats if the
    'condition is True.

    Dim a$(100)
    i% = -1
    Do
        i% = i% + 1
        If i% = 0 Then
            a(i%) = Dir$("")
        Else
            a(i%) = Dir$
        End If
    Loop While (a(i%) <> "" And i% <= 99)
    r% = SelectBox(i% & " files found",,a)
```

'This second example uses the Do While...Loop, which checks the condition and then repeats if the condition is True.

```
Dim a$(100)
i% = 0
a(i%) = Dir$("*")
Do While a(i%) <> "" And i% <= 99
    i% = i% + 1
    a(i%) = Dir$
Loop
r% = SelectBox(i% & " files found",,a)
```

'This third example uses the Do Until...Loop, which does the iteration and then checks the condition and repeats if the condition is True.

```
Dim a$(100)
i% = 0
a(i%) = Dir$("*")
Do Until a(i%) = "" Or i% = 100
    i% = i% + 1
    a(i%) = Dir$
Loop
r% = SelectBox(i% & " files found",,a)
```

'This last example uses the Do...Until Loop, which performs the iteration first, checks the condition, and repeats if the condition is True.

```
Dim a$(100)
i% = -1
Do
    i% = i% + 1
    If i% = 0 Then
        a(i%) = Dir$("*")
    Else
        a(i%) = Dir$
    End If
Loop Until (a(i%) = "" Or i% = 100)
r% = SelectBox(i% & " files found",,a)
End Sub
```

See Also For...Next (statement); While ...WEnd (statement).

Platform(s) Windows and Macintosh.

Platform Notes: Due to errors in program logic, you can inadvertently create infinite loops in your code. Under Windows, you can break out of infinite loops using Ctrl+Break.

Platform Notes: Due to errors in program logic, you can inadvertently create infinite loops in your code. On the Macintosh, you can break out of infinite loops using Command+Period.

Macintosh

DoEvents (function)

- Syntax** DoEvents[()]
- Description** Yields control to other applications, returning an Integer 0.
- Comments** This statement yields control to the operating system, allowing other applications to process mouse, keyboard, and other messages.
- If a SendKeys statement is active, this statement waits until all the keys in the queue have been processed.
- Example** See DoEvents (statement).
- See Also** DoEvents (statement).
- Platform(s)** Windows and Macintosh.

DoEvents (statement)

- Syntax** DoEvents
- Description** Yields control to other applications.
- Comments** This statement yields control to the operating system, allowing other applications to process mouse, keyboard, and other messages.
- If a SendKeys statement is active, this statement waits until all the keys in the queue have been processed.
- Examples**
- ```
'This first example shows a script that takes a long time and hogs the
'system. The subroutine explicitly yields to allow other applications
'to execute.

Sub Main()
 Open "test.txt" For Output As #1
 For i = 1 To 10000
 Print #1,"This is a test of the system and stuff."
 DoEvents
 Next i
 Close #1
End Sub
```
- 'In this second example, the DoEvents statement is used to wait until  
'the queue has been completely flushed.
- ```
Sub Main()
    AppActivate "Notepad"           'Activate Notepad.
    SendKeys "This is a test.",False 'Send some keys.
    DoEvents                         'Wait for the keys to play back.
End Sub
```
- See Also** DoEvents (function).
- Platform(s)** Windows and Macintosh.

DoKeys (statement)

Syntax `DoKeys KeyString$ [,time]`

Description Simulates the pressing of the specified keys.

Comments The DoKeys statement accepts the following parameters:

Parameter	Description
<i>KeyString\$</i>	String containing the keys to be sent. The format for <i>KeyString\$</i> is described under the SendKeys statement.
<i>time</i>	Integer specifying the number of milliseconds devoted for the output of the entire <i>KeyString\$</i> parameter. It must be within the following range:

$$0 \leq time \leq 32767$$

For example, if *time* is 5000 (5 seconds) and the *KeyString\$* parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.

Example 'This code fragment plays back the time and date into Notepad.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    id = Shell("Notepad",4)    'Run Notepad.
    AppActivate "Notepad"
    t$ = time$
    d$ = date$
    DoKeys "The time is: " & t$ & "." & crlf
    DoKeys "The date is: " & d$ & "."
End Sub
```

See Also SendKeys (statement); QueKeys (statement); QueKeyDn (statement); QueKeyUp (statement).

Platform(s) Windows.

Platform Notes: This statement uses the Windows journalizing mechanism to play keystrokes into the Windows environment.

Windows

Double (data type)

Syntax `Double`

Description A data type used to declare variables capable of holding real numbers with 15–16 digits of precision.

Comments Double variables are used to hold numbers within the following ranges:

Sign	Range
Negative	$-1.797693134862315E308 \leq \text{double} \leq -4.94066E-324$
Positive	$4.94066E-324 \leq \text{double} \leq 1.797693134862315E308$

The type-declaration character for Double is #.

Storage

Internally, doubles are 8-byte (64-bit) IEEE values. Thus, when appearing within a structure, doubles require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required.

Each Double consists of the following

- A 1-bit sign
- An 11-bit exponent
- A 53-bit significand (mantissa)

See Also Currency (data type); Date (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CDb1 (function).

Platform(s) Windows and Macintosh.

DropListBox (statement)

Syntax DropListBox *X, Y, width, height, ArrayVariable, .Identifier*

Description Creates a drop list box within a dialog box template.

Comments When the dialog box is invoked, the drop list box will be filled with the elements contained in *ArrayVariable*. Drop list boxes are similar to combo boxes, with the following exceptions:

- The list box portion of a drop list box is not opened by default. The user must open it by clicking the down arrow.
- The user cannot type into a drop list box. Only items from the list box may be selected. With combo boxes, the user can type the name of an item from the list directly or type the name of an item that is not contained within the combo box.

This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).

The DropListBox statement requires the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>ArrayVariable</i>	Single-dimensioned array used to initialize the elements of the drop list box. If this array has no dimensions, then the drop list box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension.
<i>.Identifier</i>	<i>ArrayVariable</i> can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings. Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the drop list box's selection (0 is the first item, 1 is the second, and so on). This variable can be accessed using the following syntax:

DialogVariable.Identifier

Example 'This example allows the user to choose a field name from a drop list box.

```
Sub Main()
    Dim FieldNames$(4)
    FieldNames$(0) = "Last Name"
    FieldNames$(1) = "First Name"
    FieldNames$(2) = "Zip Code"
    FieldNames$(3) = "State"
    FieldNames$(4) = "City"
    Begin Dialog FindTemplate 16,32,168,48,"Find"
        Text 8,8,37,8,"&Find what:"
        DropListBox 48,6,64,80,FieldNames,.WhichField
        OKButton 120,7,40,14
        CancelButton 120,27,40,14
    End Dialog
    Dim FindDialog As FindTemplate
    FindDialog.WhichField = 1
    Dialog FindDialog
End Sub
```

See Also CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement); PictureBox (statement).

Platform(s) Windows and Macintosh.

ebAbort (constant)

Description Returned by the MsgBox function when the Abort button is chosen.

Comments This constant is equal to 3.

Example 'This example displays a dialog box with Abort, Retry, and Ignore buttons.

```
Sub Main()
Again:
    rc% = MsgBox("Do you want to continue?",ebAbortRetryIgnore)
    If rc% = ebAbort Then
        End
    Else
        Goto Again:
    End If
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebAbortRetryIgnore (constant)

Description Used by the MsgBox statement and function.

Comments This constant is equal to 2.

Example 'This example displays a dialog box with Abort, Retry, and Ignore buttons.

```
Sub Main()
    rc% = MsgBox("Wicked disk error!",ebAbortRetryIgnore)
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebApplicationModal (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 0.

Example 'This example displays an application-modal dialog box (which is the default).

```
Sub Main()
    MsgBox "This is application-modal.",ebOKOnly Or ebApplicationModal
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebArchive (constant)

Description Bit position of a file attribute indicating that a file hasn't been backed up.

Comments This constant is equal to 32.

Example 'This example dimensions an array and fills it with filenames with the 'Archive bit set.

```
Sub Main()
    Dim s$()
    FileList s$, "**", ebArchive
    a% = SelectBox("Archived Files", "Choose one", s$)
    If a% >= 0 Then          'If a% is -1, then the user pressed Cancel.
        MsgBox "You selected Archive file: " & s$(a)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

Platform This constant only applies to Windows.

Notes:
Windows

ebBold (constant)

Description Used with the Text and TextBox statement to specify a bold font.

Comments This constant is equal to 2.

Example

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Bold Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBold
        TextBox 10,35,200,20,.Edit,, "Times New Roman",16,ebBold
        OKButton 96,110,40,14
    End Dialog
    Dim a As UserDialog
    Dialog a
End Sub
```

See Also Text (statement), TextBox (statement).

Platform(s) Windows and Macintosh.

ebBoldItalic (constant)

Description Used with the Text and TextBox statement to specify a bold-italic font.

Comments This constant is equal to 6.

Example

```
Sub Main()  
    Begin Dialog UserDialog 16,32,232,132,"Bold-Italic Font Demo"  
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBoldItalic  
        TextBox 10,35,200,20,.Edit,, "Times New Roman",16,ebBoldItalic  
        OKButton 96,110,40,14  
    End Dialog  
    Dim a As UserDialog  
    Dialog a  
End Sub
```

See Also Text (statement), TextBox (statement).

Platform(s) Windows and Macintosh.

ebBoolean (constant)

Description Number representing the type of a Boolean variant.

Comments This constant is equal to 11.

Example

```
Sub Main()  
    Dim MyVariant as variant  
    MyVariant = True  
    If VarType(MyVariant) = ebBoolean Then  
        MyVariant = 5.5  
    End If  
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebCancel (constant)

Description Returned by the MsgBox function when the Cancel button is chosen.

Comments This constant is equal to 2.

Example

```
Sub Main()  
    'Invoke MsgBox and check whether the Cancel button was pressed.  
    rc% = MsgBox("Are you sure you want to quit?",ebOKCancel)  
    If rc% = ebCancel Then  
        MsgBox "The user clicked Cancel."  
    End If  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebCritical (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 16.

Example

```
Sub Main()
    'Invoke MsgBox with Abort, Retry, and Ignore buttons and a Stop icon.
    rc% = MsgBox("Disk drive door is open.", vbAbortRetryIgnore Or
        vbCritical)
    If rc% = 3 Then
        'The user selected Abort from the dialog box.
        MsgBox "The user clicked Abort."
    End If
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebCurrency (constant)

Description Number representing the type of a Currency variant.

Comments This constant is equal to 6.

Example

```
'This example checks to see whether a variant is of type Currency.
Sub Main()
    Dim MyVariant
    If VarType(MyVariant) = vbCurrency Then
        MsgBox "Variant is Currency."
    End If
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebDataObject (constant)

Description Number representing the type of a data object variant.

Comments This constant is equal to 13.

Example 'This example checks to see whether a variable is a data object.

```
Sub Main()  
    Dim MyVariant as Variant  
    If VarType(MyVariant) = ebDataObject Then  
        MsgBox "Variant contains a data object."  
    End If  
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebError (constant)

Description Number representing the type of an error variant.

Comments This constant is equal to 10.

Example 'This example checks to see whether a variable is an error.

```
Function Div(ByVal a As Variant, ByVal b As Variant) As Variant  
    If b = 0 Then  
        Div = CVErr(20000)  
    Else  
        Div = a / b  
    End If  
End Function  
  
Sub Main()  
    Dim Result as Variant  
    Randomize  
    Result = Div(10, Random(0, 2))  
    If VarType(Result) = ebError Then  
        MsgBox "An error occurred"  
    Else  
        MsgBox "The result of the division is: " & Result  
    End If  
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebDate (constant)

Description Number representing the type of a Date variant.

Comments This constant is equal to 7.

Example

```
Sub Main()
    Dim MyVariant as Variant
    If VarType(MyVariant) = ebDate Then
        MsgBox "This variable is a Date type!"
    Else
        MsgBox "This variable is not a Date type!"
    End If
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebDefaultButton1 (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 0.

Example

```
'This example invokes MsgBox with the focus on the OK button by default.

Sub Main()
    rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or
ebDefaultButton1)
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebDefaultButton2 (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 256.

Example

```
'This example invokes MsgBox with the focus on the Cancel button by default.

Sub Main()
    rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or
ebDefaultButton2)
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebDefaultButton3 (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 512.

Example 'This example invokes MsgBox with the focus on the Ignore button by 'default.

```
Sub Main()  
    rc% = MsgBox("Disk drive door open.", vbAbortRetryIgnore Or  
        vbDefaultButton3)  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebDirectory (constant)

Description Bit position of a file attribute indicating that a file is a directory entry.

Comments This constant is equal to 16.

Example 'This example dimensions an array and fills it with directory names 'using the ebDirectory constant.

```
Sub Main()  
    Dim s$(  
    FileList s$, "c:\*", vbDirectory  
    a% = SelectBox("Directories", "Choose one:", s$)  
    If a% >= 0 Then  
        MsgBox "You selected directory: " & s(a%)  
    Else  
        MsgBox "No selection made."  
    End If  
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

ebDouble (constant)

Description Number representing the type of a Double variant.

Comments This constant is equal to 5.

Example See ebSingle (constant).

See Also MsgBox (function); MsgBox (statement); VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebEmpty (constant)

Description Number representing the type of an Empty variant.

Comments This constant is equal to 0.

Example

```
Sub Main()
Dim MyVariant as Variant
If VarType(MyVariant) = ebEmpty Then
    MsgBox "No data has been input yet!"
End If
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebExclamation (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 48.

Example

```
'This example displays a dialog box with an OK button and an
'exclamation icon.

Sub Main()
    MsgBox "Out of memory saving to disk.",ebOKOnly Or ebExclamation
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebHidden (constant)

Description Bit position of a file attribute indicating that a file is hidden.

Comments This constant is equal to 2.

Example

```
'This example dimensions an array and fills it with filenames using
'the ebHidden attribute.

Sub Main()
    Dim s$()
    FileList s$,"*",ebHidden
    If ArrayDims(s$) = 0 Then
        MsgBox "No hidden files found!"
    End
    End If
    a% = SelectBox("Hidden Files","Choose one", s$)
    If a% >= 0 Then
        MsgBox "You selected hidden file " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

ebIgnore (constant)

Description Returned by the MsgBox function when the Ignore button is chosen.

Comments This constant is equal to 5.

Example 'This example displays a critical error dialog box and sees what the user wants to do.

```
Sub Main()  
    rc% = MsgBox("Printer out of paper.",ebAbortRetryIgnore)  
    If rc% = ebIgnore Then  
        'Continue printing here.  
    End If  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebInformation (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 64.

Example 'This example displays a dialog box with the Information icon.

```
Sub Main()  
    MsgBox "You just deleted your file!",ebOKOnly Or ebInformation  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebInteger (constant)

Description Number representing the type of an Integer variant.

Comments This constant is equal to 2.

Example 'This example defines a function that returns True if a variant contains an Integer value (either a 16-bit or 32-bit Integer).

```
Function IsInteger(v As Variant) As Boolean
    If VarType(v) = ebInteger Or VarType(v) = ebLong Then
        IsInteger = True
    Else
        IsInteger = False
    End If
End Function

Sub Main()
    Dim i as Integer
    i = 123
    If IsInteger(i) then
        MsgBox "i is an Integer."
    End If
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebItalic (constant)

Description Used with the Text and TextBox statement to specify an italic font.

Comments This constant is equal to 4.

Example

```
Sub Main()
    Begin Dialog UserDialog 16,32,232,132,"Italic Font Demo"
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebItalic
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebItalic
        OKButton 96,110,40,14
    End Dialog

    Dim a As UserDialog
    Dialog a
End Sub
```

See Also Text (statement), TextBox (statement).

Platform(s) Windows and Macintosh.

ebLandscape (constant)

Description Used with the PrinterSetOrientation statement to align the paper horizontally.

Comments This constant is equal to 2.

Example 'This example sets the printer orientation to landscape.

```
Sub Main()  
    PrinterSetOrientation ebLandscape  
    MsgBox "Printer set to landscape."  
End Sub
```

See Also PrinterSetOrientation (statement); PrinterGetOrientation (function).

Platform(s) Windows.

ebLeftButton (constant)

Description Used with the QueMouseXX commands to represent the left button.

Comments This constant is equal to 1.

Example 'This example double-clicks the left mouse button.

```
Sub Main()  
    QueMouseClick ebLeftButton,1000,1875  
End Sub
```

See Also QueButtonDn (statement); QueButtonUp (statement); QueMouseClick (statement); QueMouseDblClk (statement); QueMouseDblDn (statement).

Platform(s) Windows.

ebLong (constant)

Description Number representing the type of a Long variant.

Comments This constant is equal to 3.

Example See ebInteger (constant).

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebMacintosh (constant)

Description Used with the Basic.OS property to indicate the Macintosh version of WM Basic.

Comments This constant is equal to 10.

The Basic.OS property returns this value when WM Basic is running under the Macintosh operating system

Example

```
Sub Main()  
    If Basic.OS = ebMacintosh Then MsgBox "Running on Macintosh."  
End Sub
```

See Also Basic.OS (property).

Platform(s) Windows and Macintosh.

ebMaximized (constant)

Description Used with the AppSetState and AppGetState statements to indicate a maximized window state.

Comments This constant is equal to 1.

Example 'This example minimizes the current application if it is maximized.

```
Sub Main()  
    If AppGetState = ebMaximized Then AppMinimize  
End Sub
```

See Also AppSetState (statement); AppGetState (function).

Platform(s) Windows.

ebMinimized (constant)

Description Used with the AppSetState and AppGetState statements to indicate a minimized window state.

Comments This constant is equal to 2.

Example 'This example restores the current application if it is minimized.

```
Sub Main()  
    If AppGetState = ebMinimized Then  
        AppMaximize  
    Else  
        AppMinimize  
    End If  
End Sub
```

See Also AppSetState (statement); AppGetState (function).

Platform(s) Windows.

ebNo (constant)

Description Returned by the MsgBox function when the No button is chosen.

Comments This constant is equal to 7.

Example 'This example asks a question and queries the user's response.

```
Sub Main()  
    rc% = MsgBox("Do you want to update the glossary?",ebYesNo)  
    If rc% = ebNo Then  
        MsgBox "The user clicked 'No'."          'Don't update glossary.  
    End If  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebNone (constant)

Description Bit value used to select files with no other attributes.

Comments This value can be used with the Dir\$ and FileList commands. These functions will return only files with no attributes set when used with this constant. This constant is equal to 64.

Example 'This example dimensions an array and fills it with filenames with no 'attributes set.

```
Sub Main()  
    Dim s$()  
    FileList s$,"*",ebNone  
    If ArrayDims(s$) = 0 Then  
        MsgBox "No files found without attributes!"  
    End  
End If  
a% = SelectBox("No Attributes", "Choose one", s$)  
If a% >= 0 Then  
    MsgBox "You selected file " & s(a%)  
Else  
    MsgBox "No selection made."  
End If  
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

ebNormal (constant)

Description Used to search for "normal" files.

Comments This value can be used with the Dir\$ and FileList commands and will return files with the Archive, Volume, ReadOnly, or no attributes set. It will not match files with Hidden, System, or Directory attributes. This constant is equal to 0.

Example 'This example dimensions an array and fills it with filenames with 'Normal attributes.

```
Sub Main()
    Dim s$()
    FileList s$,"*", ebNormal
    If ArrayDims(s$) = 0 Then
        MsgBox "No filesfound!"
    End
End If
a% = SelectBox("Normal Files", "Choose one", s$)
If a% >= 0 Then
    MsgBox "You selected file " & s(a%)
Else
    MsgBox "No selection made."
End If
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

ebNull (constant)

Description Number representing the type of a Null variant.

Comments This constant is equal to 1.

Example

```
Sub Main()
    Dim MyVariant
    MyVariant = Null
    If VarType(MyVariant) = ebNull Then
        MsgBox "This variant is Null"
    End If
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebObject (constant)

Description Number representing the type of an Object variant (an OLE automation object).

Comments This constant is equal to 9.

Example

```
Sub Main()
    If VarType(MyVariant) = ebObject Then
        MsgBox MyVariant.Value
    End If
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebOK (constant)

Description Returned by the MsgBox function when the OK button is chosen.

Comments This constant is equal to 1.

Example 'This example displays a dialog box that allows the user to cancel.

```
Sub Main()  
    rc% = MsgBox("Are you sure you want to exit Windows?",ebOKCancel)  
    If rc% = ebOK Then System.Exit  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebOKCancel (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 1.

Example 'This example displays a dialog box that allows the user to cancel.

```
Sub Main()  
    rc% = MsgBox("Are you sure you want to exit Windows?",ebOKCancel)  
    If rc% = ebOK Then System.Exit  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebOKOnly (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 0.

Example 'This example informs the user of what is going on (no options).

```
Sub Main()  
    MsgBox "Windows will now shut down.",ebOKOnly  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebPortrait (constant)

Description Used with the `PrinterSetOrientation` statement to align the paper vertically.

Comments This constant is equal to 1.

Example 'This example changes the printer's orientation to portrait.

```
Sub Main()  
    PrinterSetOrientation ebPortrait  
End Sub
```

See Also `PrinterSetOrientation (statement)`; `PrinterGetOrientation (function)`.

Platform(s) Windows.

ebQuestion (constant)

Description Used with the `MsgBox` statement and function.

Comments This constant is equal to 32.

Example 'This example displays a dialog box with OK and Cancel buttons and a 'question icon.

```
Sub Main()  
    rc% = MsgBox("OK to delete file?", vbOKCancel Or ebQuestion)  
End Sub
```

See Also `MsgBox (function)`; `MsgBox (statement)`.

Platform(s) Windows and Macintosh.

ebReadOnly (constant)

Description Bit position of a file attribute indicating that a file is read-only.

Comments This constant is equal to 1.

Example 'This example dimensions an array and fills it with filenames with 'ReadOnly attributes.

```
Sub Main()  
    Dim s$()  
    FileList s$, "*", ebReadOnly  
    If ArrayDims(s$) = 0 Then  
        MsgBox "No read only files found!"  
    End If  
    End If  
    a% = SelectBox("ReadOnly", "Choose one", s$)  
    If a% >= 0 Then  
        MsgBox "You selected file " & s(a%)  
    Else  
        MsgBox "No selection made."  
    End If  
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

ebRegular (constant)

Description Used with the Text and TextBox statement to specify an normal-styled font (i.e., neither bold or italic).

Comments This constant is equal to 1.

Example

```
Sub Main()  
    Begin Dialog UserDialog 16,32,232,132,"Regular Font Demo"  
        Text 10,10,200,20,"Hello, world.",,"Helv",24,ebRegular  
        TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebRegular  
        OKButton 96,110,40,14  
    End Dialog  
    Dim a As UserDialog  
    Dialog a  
End Sub
```

See Also Text (statement), TextBox (statement).

Platform(s) Windows and Macintosh.

ebRestored (constant)

Description Used with the AppSetState and AppGetState statements to indicate a normal window state.

Comments This constant is equal to 3.

Example 'This example minimizes the current application only if it is restored.

```
Sub Main()
    state% = AppGetState
    If state% = ebRestored Then
        AppMinimize
    End If
End Sub
```

See Also AppSetState (statement); AppGetState (function).

Platform(s) Windows.

ebRetry (constant)

Description Returned by the MsgBox function when the Retry button is chosen.

Comments This constant is equal to 4.

Example 'This example displays a Retry message box.

```
Sub Main()
    rc% = MsgBox("Unable to open file.",ebRetryCancel)
    If rc% = ebRetry Then
        MsgBox "User selected Retry."
    End If
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebRetryCancel (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 5.

Example 'This example invokes a dialog box with Retry and Cancel buttons.

```
Sub Main()
    rc% = MsgBox("Unable to open file.",ebRetryCancel)
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebRightButton (constant)

Description Used with the QueMouseXX commands to represent the right button.

Comments This constant is equal to 2.

Example 'This example clicks the right mouse button at 1000,1200.

```
Sub Main()  
    QueMouseClicked ebRightButton,1000,1200  
End Sub
```

See Also QueButtonDn (statement); QueButtonUp (statement); QueMouseClicked (statement); QueMouseDown (statement); QueMouseUp (statement).

Platform(s) Windows.

ebSingle (constant)

Description Number representing the type of a Single variant.

Comments This constant is equal to 4.

Example 'This example defines a function that returns True if the passed variant is a Real number.

```
Function IsReal(v As Variant) As Boolean  
    If VarType(v) = ebSingle Or VarType(v) = ebDouble Then  
        IsReal = True  
    Else  
        IsReal = False  
    End If  
End Function  
  
Sub Main()  
    Dim i as Integer  
    i = 123  
    If IsReal(i) then  
        MsgBox "i is Real."  
    End If  
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebString (constant)

Description Number representing the type of a String variant.

Comments This constant is equal to 8.

Example

```
Sub Main()  
    Dim MyVariant as variant  
    MyVariant = "This is a test."  
    If VarType(MyVariant) = ebString Then  
        MsgBox "Variant is a string."  
    End If  
End Sub
```

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebSystem (constant)

Description Bit position of a file attribute indicating that a file is a system file.

Comments This constant is equal to 4.

Example 'This example dimensions an array and fills it with filenames with 'System attributes.

```
Sub Main()
    Dim s$( )
    FileList s$,"*",ebSystem
    a% = SelectBox("System Files", "Choose one", s$)
    If a% >= 0 Then
        MsgBox "You selected file " & s(a%)
    Else
        MsgBox "No selection made."
    End If
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

Platform This constant only applies to Windows.

Notes:
Windows

ebSystemModal (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 4096.

Example

```
Sub Main()
    MsgBox "All applications are halted!",ebSystemModal
End Sub
```

See Also ebApplicationModal (constant); Constants (topic); MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebVariant (constant)

Description Number representing the type of a Variant.

Comments Currently, it is not possible for variants to use this subtype. This constant is equal to 12.

See Also VarType (function); Variant (data type).

Platform(s) Windows and Macintosh.

ebVolume (constant)

Description Bit position of a file attribute indicating that a file is the volume label.

Comments This constant is equal to 8.

Example 'This example dimensions an array and fills it with filenames with
'Volume attributes.

```
Sub Main()  
    Dim s$()  
    FileList s$, "*", ebVolume  
    If ArrayDims(s$) > 0 Then  
        MsgBox "The volume name is: " & s(1)  
    Else  
        MsgBox "No volumes found."  
    End If  
End Sub
```

See Also Dir, Dir\$ (functions); FileList (statement); SetAttr (statement); GetAttr (function); FileAttr (function).

Platform(s) Windows and Macintosh.

Platform This constant only applies to Windows.

Notes:
Windows

ebWin16 (constant)

Description Used with the Basic.OS property to indicate the 16-bit Windows version of WM Basic.

Comments This constant is equal to 0.

The Basic.OS property returns this value when WM Basic is running under the Windows 3.1 operating system

Example Sub Main()
 If Basic.OS = **ebWin16** Then MsgBox "Running under Windows 3.1."
End Sub

See Also Basic.OS (property).

Platform(s) Windows and Macintosh.

ebWindows (constant)

Description Used with the AppType function to indicate a Windows application.

Comments This constant is equal to 2.

Example 'This example determines whether a Windows application was selected.

```
Sub Main()
    s$ = OpenFilename$("Run","Programs:*.exe")
    If s$ <> "" Then
        If FileType(s$) = ebWindows Then
            MsgBox "You selected a Windows .exe file."
        End If
    End If
End Sub
```

See Also AppGetType (function); AppFileType (function).

Platform(s) Windows.

ebYes (constant)

Description Returned by the MsgBox function when the Yes button is chosen.

Comments This constant is equal to 6.

Example 'This example queries the user for a response.

```
Sub Main()
    rc% = MsgBox("Overwrite file?",ebYesNoCancel)
    If rc% = ebYes Then
        MsgBox "You elected to overwrite the file."
    End If
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebYesNo (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 4.

Example 'This example displays a dialog box with Yes and No buttons.

```
Sub Main()
    rc% = MsgBox("Are you sure you want to remove all  
formatting?",ebYesNo)
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

ebYesNoCancel (constant)

Description Used with the MsgBox statement and function.

Comments This constant is equal to 3.

Example 'This example displays a dialog box with Yes, No, and Cancel buttons.

```
Sub Main()  
    rc% = MsgBox("Format drive C:?",ebYesNoCancel)  
    If rc% = ebYes Then  
        MsgBox "The user chose Yes."  
    End If  
End Sub
```

See Also MsgBox (function); MsgBox (statement).

Platform(s) Windows and Macintosh.

EditEnabled (function)

Syntax EditEnabled(*name\$* | *id*)

Description Returns True if the given text box is enabled within the active window or dialog box; returns False otherwise.

Comments The EditEnabled function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the text box. The name of a text box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a text box.
<i>id</i>	Integer specifying the ID of the text box. A runtime error is generated if a text box control with the given name or ID cannot be found within the active window. If enabled, the text box can be given the focus using the ActivateControl statement.
Note: The EditEnabled function is used to determine whether a text box is enabled in another application's dialog box. Use the DlgEnable function in dynamic dialog boxes.	

Example 'This example adjusts the left margin if this control is enabled.

```
Sub Main()
    Menu "Format.Paragraph"
    If EditEnabled("Left:") Then
        SetEditText "Left:", "5 pt"
    End If
End Sub
```

See Also EditExists (function); GetEditText\$ (function); SetEditText (statement).

Platform(s) Windows.

EditExists (function)

Syntax EditExists(*name\$* | *id*)

Description Returns True if the given text box exists within the active window or dialog box; returns False otherwise.

Comments The EditExists function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the text box. The name of a text box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a text box.
<i>id</i>	Integer specifying the ID of the text box. A runtime error is generated if a text box control with the given name or ID cannot be found within the active window. If there is no active window, False will be returned.
Note: The EditExists function is used to determine whether a text box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example 'This example adjusts the left margin if this control exists and is 'enabled.

```
Sub Main()
    Menu "Format.Paragraph"
    If EditExists("Left:") Then
        If EditEnabled("Left:") Then
            SetEditText "Left:", "5 pt"
        End If
    End If
End Sub
```

See Also EditEnabled (function); GetEditText\$ (function); SetEditText (statement).

Platform(s) Windows.

Empty (constant)

Description Constant representing a variant of type 0.

Comments The Empty value has special meaning indicating that a Variant is uninitialized.

When Empty is assigned to numbers, the value 0 is assigned. When Empty is assigned to a String, the string is assigned a zero-length string.

Example

```
Sub Main()  
    Dim a As Variant  
    a = Empty  
End Sub
```

See Also Null (constant); Variant (data type); VarType (function).

Platform(s) Windows and Macintosh.

End (statement)

Syntax End

Description Terminates execution of the current script, closing all open files.

Example 'This example uses the End statement to stop execution.

```
Sub Main()  
    MsgBox "The next line will terminate the script."  
    End  
End Sub
```

See Also Close (statement); Stop (statement); Exit For (statement); Exit Do (statement); Exit Function (statement); Exit Sub (function).

Platform(s) Windows and Macintosh.

Environ, Environ\$ (functions)

Syntax Environ[\$](variable\$ | VariableNumber)

Description Returns the value of the specified environment variable.

Comments Environ\$ returns a String, whereas Environ returns a String variant.

If *variable\$* is specified, then this function looks for that *variable\$* in the environment. If the *variable\$* name cannot be found, then a zero-length string is returned.

If *VariableNumber* is specified, then this function looks for the *N*th variable within the environment (the first variable being number 1). If there is no such environment variable, then a zero-length string is returned. Otherwise, the entire entry from the environment is returned in the following format:

variable = value

Example 'This example looks for the DOS Comspec variable and displays the 'value in a dialog box.

```
Sub Main()
    Dim a$(1)
    a$(1) = Environ$( "COMSPEC" )
    MsgBox "The DOS Comspec variable is set to: " & a$(1)
End Sub
```

See Also Command, Command\$ (functions).

Platform(s) Windows and Macintosh.

EOF (function)

Syntax EOF(*filenumber*)

Description Returns True if the end-of-file has been reached for the given file; returns False otherwise.

Comments The *filenumber* parameter is an Integer used by WM Basic to refer to the open file—the number passed to the Open statement.

With sequential files, EOF returns True when the end of the file has been reached (i.e., the next file read command will result in a runtime error).

With Random or Binary files, EOF returns True after an attempt has been made to read beyond the end of the file. Thus, EOF will only return True when Get was unable to read the entire record.

Example 'This example opens the autoexec.bat file and reads lines from the 'file until the end-of-file is reached.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim s$
    Open "c:\autoexec.bat" For Input As #1
    Do While Not EOF(1)
        Input #1,s$
    Loop
    Close
    MsgBox "The last line was:" & crlf & s$
End Sub
```

See Also Open (statement); LOF (function).

Platform(s) Windows and Macintosh.

Eqv (operator)

Syntax expression1 Eqv expression2

Description Performs a logical or binary equivalence on two expressions.

Comments If both expressions are either Boolean, Boolean variants, or Null variants, then a logical equivalence is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
False	True	False
False	False	True

If either expression is Null, then Null is returned.

Binary Equivalence

If the two expressions are Integer, then a binary equivalence is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary equivalence is then performed, returning a Long result.

Binary equivalence forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions, according to the following table:

1	Eqv1	=	1	Example:
0	Eqv1	=	0	5 01101001
1	Eqv0	=	0	<u>6 10101010</u>
0	Eqv0	=	1	Eqv00101000

Example 'This example assigns False to A, performs some equivalent operations, 'and displays a dialog box with the result. Since A is equivalent to 'False, and False is equivalent to 0, and by definition, A = 0, then 'the dialog box will display "A is False."

```
Sub Main()  
    a = False  
    If ((a Eqv False) And (False Eqv 0) And (a = 0)) Then  
        MsgBox "a is False."  
    Else  
        MsgBox "a is True."  
    End If  
End Sub
```

See Also Operator Precedence (topic); Or (operator); Xor (operator); Imp (operator); And (operator).

Platform(s) Windows and Macintosh.

Erase (statement)

Syntax Erase array1 [,array2]...

Description Erases the elements of the specified arrays.

Comments For dynamic arrays, the elements are erased, and the array is redimensioned to have no dimensions (and therefore no elements). For fixed arrays, only the elements are erased; the array dimensions are not changed.

After a dynamic array is erased, the array will contain no elements and no dimensions. Thus, before the array can be used by your program, the dimensions must be reestablished using the `Redim` statement.

Up to 32 parameters can be specified with the `Erase` statement.

The meaning of erasing an array element depends on the type of the element being erased:

Element Type	What Erase Does to That Element
Integer	Sets the element to 0.
Boolean	Sets the element to <code>False</code> .
Long	Sets the element to 0.
Double	Sets the element to 0.0.
Date	Sets the element to December 30, 1899.
Single	Sets the element to 0.0.
String (variable-length)	Frees the string, then sets the element to a zero-length string.
String (fixed-length)	Sets every character of each element to zero (<code>Chr\$(0)</code>).
Object	Decrements the reference count and sets the element to <code>Nothing</code> .
Variant	Sets the element to <code>Empty</code> .
User-defined type	Sets each structure element as a separate variable.

Example 'This example puts a value into an array and displays it.
'Then it erases the value and displays it again.

```
Sub Main()  
    Dim a$(10)           'Declare an array.  
    a$(1) = Dir$("*")     'Fill element 1 with a filename.  
    MsgBox "Array before Erase: " & a$(1)      'Display element 1.  
    Erase a$              'Erase all elements in the array.  
    MsgBox "Array after Erase: " & a$(1)       'Display element 1 again  
    (should be erased).  
End Sub
```

See Also Redim (statement); Arrays (topic).

Platform(s) Windows and Macintosh.

Erl (function)

Syntax `Erl[()]`

Description Returns the line number of the most recent error.

Comments The first line of the script is 1, the second line is 2, and so on.
The internal value of `Err` is reset to 0 with any of the following statements:
`Resume`, `Exit Sub`, `Exit Function`. Thus, if you want to use this value outside an error handler, you must assign it to a variable.

Example 'This example generates an error and then determines the line
'on which the error occurred.

```
Sub Main()  
    Dim i As Integer  
    On Error Goto Trap1  
    i = 32767 'Generate an error--overflow.  
    i = i + 1  
    Exit Sub  
Trap1:  
    MsgBox "Error on line: " & Err  
    Exit Sub 'Reset the error handler.  
End Sub
```

See Also `Err` (function); `Error`, `Error$` (functions); Error Handling (topic).

Platform(s) Windows and Macintosh.

Err (function)

Syntax `Err[()]`

Description Returns an `Integer` representing the error that caused the current error trap.

Comments The `Err` function can only be used while within an error trap.
The internal value of `Err` is reset to 0 with any of the following statements:
`Resume`, `Exit Sub`, `Exit Function`. Thus, if you want to use this value outside an error handler, you must assign it to a variable.

Example 'This example forces error 10, with a subsequent transfer to
'the TestError label. TestError tests the error and, if not
'error 55, resets Err to 999 (user-defined error) and returns to
'the Main subroutine.

```
Sub Main()  
    On Error Goto TestError  
    Error 10  
    MsgBox "The returned error is: '" & Err() & "' - '" & Error$ & "'" & vbCrLf  
    Exit Sub  
  
TestError:  
    If Err = 55 Then          'File already open.  
        MsgBox "Cannot copy an open file. Close it and try again."  
    Else  
        MsgBox "Error '" & Err & "' has occurred!"  
        Err = 999  
    End If  
    Resume Next  
End Sub
```

See Also Err (function); Error, Error\$ (functions); Error Handling (topic).

Platform(s) Windows and Macintosh.

Err (statement)

Syntax Err = *value*

Description Sets the value returned by the Err function to a specific Integer value.

Comments Only positive values less than or equal to 32767 can be used.

Setting *value* to _1 has the side effect of resetting the error state. This allows you to perform error trapping within an error handler. The ability to reset the error handler while within an error trap is not standard Basic. Normally, the error handler is reset only with the Resume, Exit Sub, or Exit Function statement.

Example 'This example forces error 10, with a subsequent transfer to 'the TestError label. TestError tests the error and, if not 'error 55, resets Err to 999 (user-defined error) and returns to 'the Main subroutine.

```
Sub Main()
    On Error Goto TestError
    Error 10
    MsgBox "The returned error is: '" & Err() & " - '" & Error$ & "'"
    Exit Sub

TestError:
    If Err = 55 Then          'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "Error '" & Err & "' has occurred."
        Err = 999
    End If
    Resume Next
End Sub
```

See Also Error (statement); Error Handling (topic).

Platform(s) Windows and Macintosh.

Error (statement)

Syntax Error *errornumber*

Description Simulates the occurrence of the given runtime error.

Comments The *errornumber* parameter is any Integer containing either a built-in error number or a user-defined error number. The Err function can be used within the error trap handler to determine the value of the error.

Example 'This example forces error 10, with a subsequent transfer to 'the TestError label. TestError tests the error and, if not 'error 55, resets Err to 999 (user-defined error) and returns to 'the Main subroutine.

```
Sub Main()
    On Error Goto TestError
    Error 10
    MsgBox "The returned error is: '" & Err() & " - '" & Error$ & "'"
    Exit Sub

TestError:
    If Err = 55 Then          'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "Error '" & Err & "' has occurred."
        Err = 999
    End If
    Resume Next
End Sub
```

See Also `Err` (statement); Error Handling (topic).

Platform(s) Windows and Macintosh.

Error Handling (topic)

Error Handlers

WM Basic supports nested error handlers. When an error occurs within a subroutine, WM Basic checks for an `On Error` handler within the currently executing subroutine or function. An error handler is defined as follows:

```
Sub foo()  
    On Error Goto catch  
    'Do something here.  
    Exit Sub  
  
catch:  
    'Handle error here.  
End Sub
```

Error handlers have a life local to the procedure in which they are defined. The error is reset when (1) another `On Error` statement is encountered, (2) an error occurs, or (3) the procedure returns.

Cascading Errors

If a runtime error occurs and no `On Error` handler is defined within the currently executing procedure, then WM Basic returns to the calling procedure and executes the error handler there. This process repeats until a procedure is found that contains an error handler or until there are no more procedures. If an error is not trapped or if an error occurs within the error handler, then WM Basic displays an error message, halting execution of the script.

Once an error handler has control, it must address the condition that caused the error and resume execution with the `Resume` statement. This statement resets the error handler, transferring execution to an appropriate place within the current procedure. An error is displayed if a procedure exits without first executing `Resume` or `Exit`.

Visual Basic Compatibility

Where possible, WM Basic has the same error numbers and error messages as Visual Basic. This is useful for porting scripts between environments.

Handling errors in WM Basic involves querying the error number or error text using the `Error$` or `Err` function. Since this is the only way to handle errors in WM Basic, compatibility with Visual Basic's error numbers and messages is essential.

WM Basic errors fall into three categories:

1. **Visual Basic-compatible errors:** These errors, numbered between 0 and 799, are numbered and named according to the errors supported by Visual Basic.
2. **WM Basic errors:** These errors, numbered from 800 to 999, are unique to WM Basic.
3. **User-defined errors:** These errors, equal to or greater than 1,000, are available for use by extensions or by the script itself.

You can intercept trappable errors using WM Basic's `On Error` construct. Almost all errors in WM Basic are trappable except for various system errors.

Error, Error\$ (functions)

Syntax `Error[$][(errornumber)]`

Description Returns a `String` containing the text corresponding to the given error number or the most recent error.

Comments `Error$` returns a `String`, whereas `Error` returns a `String` variant.

The *errornumber* parameter is an `Integer` containing the number of the error message to retrieve. If this parameter is omitted, then the function returns the text corresponding to the most recent runtime error. If no runtime error has occurred, then a zero-length string is returned.

If the `Error` statement was used to generate a user-defined runtime error, then this function will return a zero-length string (`" "`).

Example 'This example forces error 10, with a subsequent transfer to the `TestError` label. `TestError` tests the error and, if not error 55, resets `Err` to 999 (user-defined error) and returns to the `Main` subroutine.

```
Sub Main()
    On Error Goto TestError
    Error 10
    MsgBox "The returned error is: '" & Err() & "' - '" & Error$ & "'"
    Exit Sub

TestError:
    If Err = 55 Then          'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "Error '" & Err & "' has occurred."
        Err = 999
    End If
    Resume Next
End Sub
```

See Also Erl (function); Err (function); Error Handling (topic).

Platform(s) Windows and Macintosh.

Exit Do (statement)

Syntax Exit Do

Description Causes execution to continue on the statement following the Loop clause.

Comments This statement can only appear within a Do . . . Loop statement.

Example 'This example will load an array with directory entries unless there
'are more than ten entries--in which case, the Exit Do terminates
'the loop.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$(5)
    Do
        i% = i% + 1
        If i% = 1 Then
            a(i%) = Dir$("*")
        Else
            a(i%) = Dir$
        End If
        If i% >= 10 Then Exit Do
    Loop While (a(i%) <> "")

    If i% = 10 Then
        MsgBox i% & " entries processed!"
    Else
        MsgBox "Less than " & i% & " entries processed!"
    End If
End Sub
```

See Also Stop (statement); Exit For (statement); Exit Function (statement); Exit Sub (statement); End (function); Do . . . Loop (statement).

Platform(s) Windows and Macintosh.

Exit For (statement)

Syntax Exit For

Description Causes execution to exit the innermost For loop, continuing execution on the line following the Next statement.

Comments This statement can only appear within a For . . . Next block.

Example 'This example will fill an array with directory entries until a null 'entry is encountered or 100 entries have been processed--at which 'time, the loop is terminated by an Exit For statement. The dialog box 'displays a count of files found and then some entries from the array.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$(100)
    For i = 1 To 100
        If i = 1 Then
            a$(i) = Dir$("*")
        Else
            a$(i) = Dir$
        End If
        If (a$(i) = "") Or (i >= 100) Then Exit For
    Next i
    msg = "There are " & i & " files found." & crlf
    MsgBox msg & a$(1) & crlf & a$(2) & crlf & a$(3) & crlf & a$(10)
End Sub
```

See Also Stop (statement); Exit Do (statement); Exit Function (statement); Exit Sub (statement); End (statement); For...Next (statement).

Platform(s) Windows and Macintosh.

Exit Function (statement)

Syntax Exit Function

Description Causes execution to exit the current function, continuing execution on the statement following the call to this function.

Comments This statement can only appear within a function.

Example 'This function displays a message and then terminates with Exit 'Function.

```
Function Test_Exit() As Integer
    MsgBox "Testing function exit, returning to Main()."
    Test_Exit = 0
    Exit Function
    MsgBox "This line should never execute."
End Function

Sub Main()
    a% = Test_Exit()
    MsgBox "This is the last line of Main()."
End Sub
```

See Also Stop (statement); Exit For (statement); Exit Do (statement); Exit Sub (statement); End (statement); Function...End Function (statement).

Platform(s) Windows and Macintosh.

Exit Sub (statement)

Syntax Exit Sub

Description Causes execution to exit the current subroutine, continuing execution on the statement following the call to this subroutine.

Comments This statement can appear anywhere within a subroutine. It cannot appear within a function.

Example 'This example displays a dialog box and then exits. The last line 'should never execute because of the Exit Sub statement.

```
Sub Main()  
    MsgBox "Terminating Main()."  
    Exit Sub  
    MsgBox "Still here in Main()."  
End Sub
```

See Also Stop (statement); Exit For (statement); Exit Do (statement); Exit Function (statement); End (function); Sub...End Sub (statement).

Platform(s) Windows and Macintosh.

Exp (function)

Syntax Exp(*value*)

Description Returns the value of *e* raised to the power of *value*.

Comments The *value* parameter is a Double within the following range:

$0 \leq \text{value} \leq 709.782712893$.

A runtime error is generated if *value* is out of the range specified above.

The value of *e* is 2.71828.

Example 'This example assigns a to *e* raised to the 12.4 power and displays it 'in a dialog box.

```
Sub Main()  
    a# = Exp(12.40)  
    MsgBox "e to the 12.4 power is: " & a#  
End Sub
```

See Also Log (function).

Platform(s) Windows and Macintosh.

Expression Evaluation (topic)

WM Basic allows expressions to involve data of different types. When this occurs, the two arguments are converted to be of the same type by promoting the less precise operand to the same type as the more precise operand. For example, WM Basic will promote the value of `i%` to a `Double` in the following expression:

```
result# = i% * d#
```

In some cases, the data type to which each operand is promoted is different than that of the most precise operand. This is dependent on the operator and the data types of the two operands and is noted in the description of each operator.

If an operation is performed between a numeric expression and a `String` expression, then the `String` expression is usually converted to be of the same type as the numeric expression. For example, the following expression converts the `String` expression to an `Integer` before performing the multiplication:

```
result = 10 * "2"      'Result is equal to 20.
```

There are exceptions to this rule as noted in the description of the individual operators.

Type Coercion

WM Basic performs numeric type conversion automatically. Automatic conversions sometimes result in overflow errors, as shown in the following example:

```
d# = 45354
i% = d#
```

In this example, an overflow error is generated because the value contained in `d#` is larger than the maximum size of an `Integer`.

Rounding

When floating-point values (`Single` or `Double`) are converted to integer values (`Integer` or `Long`), the fractional part of the floating-point number is lost, rounding to the nearest integer value. WM Basic uses Baker's rounding:

- If the fractional part is larger than .5, the number is rounded up.
- If the fractional part is smaller than .5, the number is rounded down.
- If the fractional part is equal to .5, then the number is rounded up if it is odd and down if it is even.

The following table shows sample values before and after rounding:

Before Rounding	After Rounding to Whole Number
-----------------	--------------------------------

2.1	2
4.6	5
2.5	2
3.5	4

Default Properties

When an OLE object variable or an `Object` variant is used with numerical operators such as addition or subtraction, then the default property of that object is automatically retrieved. For example, consider the following:

```
Dim Excel As Object
Set Excel = GetObject(,"Excel.Application")
MsgBox "This application is " & Excel
```

The above example displays `This application is Microsoft Excel` in a dialog box. When the variable `Excel` is used within the expression, the default property is automatically retrieved, which, in this case, is the string `Microsoft Excel`. Considering that the default property of the `Excel` object is `.Value`, then the following two statements are equivalent:

```
MsgBox "This application is " & Excel
MsgBox "This application is " & Excel.Value
```

False (constant)

Description Boolean constant whose value is False.

Comments Used in conditionals and Boolean expressions.

Example 'This example assigns False to A, performs some equivalent operations, 'and displays a dialog box with the result. Since A is equivalent to 'False, and False is equivalent to 0, and by definition, A = 0, then 'the dialog box will display "A is False."

```
Sub Main()
    a = False
    If ((a = False) And (False Eqv 0) And (a = 0)) Then
        MsgBox "a is False."
    Else
        MsgBox "a is True."
    End If
End Sub
```

See Also True (constant); Constants (topic); Boolean (data type).

Platform(s) Windows and Macintosh.

FileAttr (function)

Syntax FileAttr(*filenumber*, *attribute*)

Description Returns an Integer specifying the file mode (if *attribute* is 1) or the operating system file handle (if *attribute* is 2).

Comments The FileAttr function takes the following parameters:

Parameter	Description										
<i>filenumber</i>	Integer value used by WM Basic to refer to the open file—the number passed to the Open statement.										
<i>attribute</i>	Integer specifying the type of value to be returned. If <i>attribute</i> is 1, then one of the following values is returned: <table> <tr> <td>1</td><td>Input</td></tr> <tr> <td>2</td><td>Output</td></tr> <tr> <td>4</td><td>Random</td></tr> <tr> <td>8</td><td>Append</td></tr> <tr> <td>32</td><td>Binary</td></tr> </table>	1	Input	2	Output	4	Random	8	Append	32	Binary
1	Input										
2	Output										
4	Random										
8	Append										
32	Binary										

If *attribute* is 2, then the operating system file handle is returned. On most systems, this is a special Integer value identifying the file.

Example 'This example opens a file for input, reads the file attributes, and 'determines the file mode for which it was opened. The result is 'displayed in a dialog box.

```
Sub Main()  
  Open "c:\autoexec.bat" For Input As #1  
  a% = FileAttr(1,1)  
  Select Case a%  
    Case 1  
      MsgBox "Opened for input."  
    Case 2  
      MsgBox "Opened for output."  
    Case 4  
      MsgBox "Opened for random."  
    Case 8  
      MsgBox "Opened for append."  
    Case 32  
      MsgBox "Opened for binary."  
    Case Else  
      MsgBox "Unknown file mode."  
  End Select  
  a% = FileAttr(1,2)  
  MsgBox "File handle is: " & a%  
  Close  
End Sub
```

See Also FileLen (function); GetAttr (function); FileType (function); FileExists (function); Open (statement); SetAttr (statement).

Platform(s) Windows and Macintosh.

FileCopy (statement)

Syntax FileCopy *source\$*, *destination\$*

Description Copies a *source\$* file to a *destination\$* file.

Comments The FileCopy function takes the following parameters:

Parameter	Description
<i>source\$</i>	String containing the name of a single file to copy. The <i>source\$</i> parameter cannot contain wildcards (? or *) but may contain path information.
<i>destination\$</i>	String containing a single, unique destination file, which may contain a drive and path specification. The file will be copied and renamed if the <i>source\$</i> and <i>destination\$</i> filenames are not the same. Some platforms do not support drive letters and may not support dots to indicate current and parent directories.

Example 'This example copies the autoexec.bat file to "autoexec.sav", then
'opens the copied file and tries to copy it again--which generates an
'error.

```
Sub Main()
    On Error Goto ErrHandler
    FileCopy "c:\autoexec.bat", "c:\autoexec.sav"
    Open "c:\autoexec.sav" For Input As # 1
    FileCopy "c:\autoexec.sav", "c:\autoexec.sv2"
    Close
    Exit Sub

ErrHandler:
    If Err = 55 Then          'File already open.
        MsgBox "Cannot copy an open file. Close it and try again."
    Else
        MsgBox "An unspecified file copy error has occurred."
    End If
    Resume Next
End Sub
```

See Also Kill (statement); Name (statement).

Platform(s) Windows and Macintosh.

FileDateTime (function)

Syntax FileDateTime(*filename\$*)

Description Returns a Date variant representing the date and time of the last modification of a file.

Comments This function retrieves the date and time of the last modification of the file specified by *filename\$* (wildcards are not allowed). A runtime error results if the file does not exist. The value returned can be used with the date/time functions (i.e., Year, Month, Day, Weekday, Minute, Second, Hour) to extract the individual elements.

Example 'This example gets the file date/time of the autoexec.bat file and
'displays it in a dialog box.

```
Sub Main()
    If FileExists("c:\autoexec.bat") Then
        a# = FileDateTime("c:\autoexec.bat")
        MsgBox "The date/time information for the file is: " & Year(a#)
        & "-" & Month(a#) & "-" & Day(a#)
    Else
        MsgBox "The file does not exist."
    End If
End Sub
```

See Also FileLen (function); GetAttr (function); FileType (function); FileAttr (function); FileExists (function).

Platform(s) Windows and Macintosh.

FileDirs (statement)

Syntax FileDirs *array()* [, *dirspec\$*]

Description Fills a String or Variant *array* with directory names from disk.

Comments The FileDirs statement takes the following parameters:

Parameter	Description
<i>array()</i>	<p>Either a zero- or a one-dimensional array of strings or variants. The array can be either dynamic or fixed.</p> <p>If <i>array()</i> is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound, UBound, and ArrayDims functions to determine the number and size of the new array's dimensions.</p> <p>If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.</p>
<i>dirspec\$</i>	<p>String containing the file search mask, such as:</p> <pre>t*. c:*.*</pre> <p>If this parameter is omitted, then * is used, which fills the array with all the subdirectory names within the current directory.</p>
Example	<p>'This example fills an array with directory entries and displays the 'first one.</p> <pre>Sub Main() Dim a\$() FileDirs a\$, "c:*.*" MsgBox "The first directory is: " & a\$(0) End Sub</pre>
See Also	FileList (statement); Dir, Dir\$ (functions); CurDir, CurDir\$ (functions); ChDir (statement).

Platform(s) Windows and Macintosh.

FileExists (function)

Syntax FileExists(*filename\$*)

Description Returns True if *filename\$* exists; returns False otherwise.

Comments This function determines whether a given *filename\$* is valid.

This function will return False if *filename\$* specifies a subdirectory.

Example 'This example checks to see whether there is an autoexec.bat file in the root directory of the C drive, then displays either its date and time of creation or the fact that it does not exist.

```
Sub Main()
    If FileExists("c:\autoexec.bat") Then
        MsgBox "This file exists!"
    Else
        MsgBox "File does not exist."
    End If
End Sub
```

See Also FileLen (function); GetAttr (function); FileType (function); FileAttr (function); FileParse\$ (function).

Platform(s) Windows and Macintosh.

FileLen (function)

Syntax FileLen(*filename\$*)

Description Returns a Long representing the length of *filename\$* in bytes.

Comments This function is used in place of the LOF function to retrieve the length of a file without first opening the file. A runtime error results if the file does not exist.

Example 'This example checks to see whether there is a c:\autoexec.bat file and, if there is, displays the length of the file.

```
Sub Main()
    If (FileExists("c:\autoexec.bat") And (FileLen("c:\autoexec.bat")
    <> 0)) Then
        b% = FileLen("c:\autoexec.bat")
        MsgBox "The length of autoexec.bat is: " & b%
    Else
        MsgBox "File does not exist."
    End If
End Sub
```

See Also GetAttr (function); FileType (function); FileAttr (function); FileParse\$ (function); FileExists (function); Loc (function).

Platform(s) Windows and Macintosh.

FileList (statement)

Syntax FileList array() [, [*filespec\$*] [, [*include_attr*] [, *exclude_attr*]]]

Description Fills a `String` or `Variant` array with filenames from disk.

Comments The `FileList` function takes the following parameters:

Parameter	Description
<i>array()</i>	<p>Either a zero- or a one-dimensional array of strings or variants. The array can be either dynamic or fixed.</p> <p>If <i>array()</i> is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the <code>LBound</code>, <code>UBound</code>, and <code>ArrayDims</code> functions to determine the number and size of the new array's dimensions.</p> <p>If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for <code>String</code> arrays) or <code>Empty</code> (for <code>Variant</code> arrays). A runtime error results if the array is too small to hold the new elements.</p>
<i>filespec\$</i>	<p><code>String</code> specifying which filenames are to be included in the list.</p> <p>The <i>filespec\$</i> parameter can include wildcards, such as <code>*</code> and <code>?</code>. If this parameter is omitted, then <code>*</code> is used.</p>
<i>include_attr</i>	<p><code>Integer</code> specifying attributes of files you want included in the list. It can be any combination of the attributes listed below.</p> <p>If this parameter is omitted, then the value 97 is used (<code>ebReadOnly</code> Or <code>ebArchive</code> Or <code>ebNone</code>).</p>
<i>exclude_attr</i>	<p><code>Integer</code> specifying attributes of files you want excluded from the list. It can be any combination of the attributes listed below.</p> <p>If this parameter is omitted, then the value 18 is used (<code>ebHidden</code> Or <code>ebDirectory</code>). In other words, hidden files and subdirectories are excluded from the list.</p>

Wildcards

The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complete searching patterns. The following table shows some examples:

This Pattern	Matches These Files	Doesn't Match These Files
S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT
C*T.TXT	CAT.TXT	CAP.TXT ACATS.TXT
C*T	CAT	CAT.DOC CAP.TXT
C?T	CAT CUT	CAT.TXT CAPIT CT
*	(All files)	

File Attributes

These numbers can be any combination of the following:

Constant	Value	Includes
ebNormal	0	Read-only, archive, subdir, none
ebReadOnly	1	Read-only files
ebHidden	2	Hidden files
ebSystem	4	System files
ebVolume	8	Volume label
ebDirectory	16	DOS subdirectories
ebArchive	32	Files that have changed since the last backup
ebNone	64	Files with no attributes

Example 'This example fills an array a with the directory of the current drive
'for all files that have normal or no attributes and excludes those
'with system attributes. The dialog box displays four filenames from
'the array.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$()
    FileList a$,"*.*", (ebNormal + ebNone), ebSystem
    If ArrayDims(a$) > 0 Then
        MsgBox a$(1) & crlf & a$(2) & crlf & a$(3) & crlf & a$(4)
    Else
        MsgBox "No files found."
    End If
End Sub
```

See Also FileDirs (statement); Dir, Dir\$ (functions).

Platform(s) Windows and Macintosh.

Platform Notice that WM Basic's filename matching is different than DOS's. The pattern

Notes: "*.*" under DOS matches all files. With WM Basic, this pattern matches only
Windows files that have file extensions.

FileParse\$ (function)

Syntax FileParse\$(filename\$[, operation])

Description Returns a String containing a portion of *filename\$* such as the path, drive, or file extension.

Comments The *filename\$* parameter can specify any valid filename (it does not have to exist). For example:

```
..\test.dat
c:\sheets\test.dat
test.dat
```

A runtime error is generated if *filename\$* is a zero-length string.

The optional *operation* parameter is an Integer specifying which portion of the *filename\$* to extract. It can be any of the following values.

Value	Meaning	Example
0	Full name	c:\sheets\test.dat
1	Drive	c
2	Path	c:\sheets
3	Name	test.dat
4	Root	test
5	Extension	dat

If *operation* is not specified, then the full name is returned. A runtime error will result if *operation* is not one of the above values.

A runtime error results if *filename\$* is empty.

Example 'This example parses the file string "c:\testsub\autoexec.bat" into its 'component parts and displays them in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim a$(6)
    For i = 1 To 5
        a$(i) = FileParse$("c:\testsub\autoexec.bat", i - 1)
    Next i
    MsgBox a$(1) & crlf & a$(2) & crlf & a$(3) & crlf & a$(4) & crlf &
a$(5)
End Sub
```

See Also FileLen (function); GetAttr (function); FileType (function); FileAttr (function); FileExists (function).

Platform(s) Windows and Macintosh.

Platform Notes: On systems that do not support drive letters, operation 1 will return a zero-length string.

The path separator is different on different platforms. Under Windows, the backslash and forward slash can be used interchangeably. For example, "c:\test.dat" is the same as "c:/test.dat".

Platform On the Macintosh, all characters are valid within filenames except colons, which are seen as path separators.

Notes:
Macintosh

FileType (function)

Syntax `FileType(filename$)`

Description Returns the type of the specified file.

Comments One of the following Integer constants is returned:

Constant	Value	Description
ebDos	1	A DOS executable file (exe files only; com files are not recognized).
ebWindows	2	A Windows executable file.

If one of the above values is not returned, then the file type is unknown.

Example 'This example looks at c:\windows\winfile.exe and determines whether 'it is a DOS or a Windows file. The result is displayed in a dialog 'box.

```
Sub Main()  
  a = FileType("c:\windows\winfile.exe")  
  If a = ebDos Then  
    MsgBox "This is a DOS file."  
  Else  
    MsgBox "This is a Windows file of type '" & a & '"  
  End If  
End Sub
```

See Also FileLen (function); GetAttr (function); FileAttr (function); FileExists (function).

Platform(s) Windows.

Platform Currently, only files with a ".exe" extension can be used with this function.

Notes: Files with a ".com" or ".bat" extension will return 3 (unknown).
Windows

Fix (function)

Syntax `Fix(number)`

Description Returns the integer part of *number*.

Comments This function returns the integer part of the given value by removing the fractional part. The sign is preserved.

The `Fix` function returns the same type as *number*, with the following exceptions:

- If *number* is `Empty`, then an `Integer` variant of value 0 is returned.
- If *number* is a `String`, then a `Double` variant is returned.
- If *number* contains no valid data, then a `Null` variant is returned.

Example 'This example returns the fixed part of a number and assigns it to b, 'then displays the result in a dialog box.

```
Sub Main()
    a# = -19923.45
    b% = Fix(a#)
    MsgBox "The fixed portion of -19923.45 is: " & b%
End Sub
```

See Also `Int` (function); `CInt` (function).

Platform(s) Windows and Macintosh.

For...Next (statement)

Syntax For *counter* = *start* To *end* [Step *increment*]
 [*statements*]
 [Exit For]
 [*statements*]
 Next [*counter* [,*nextcounter*] ...]

Description Repeats a block of statements a specified number of times, incrementing a loop counter by a given increment each time through the loop.

Comments The `For` statement takes the following parameters:

Parameter	Description
<i>counter</i>	Name of a numeric variable. Variables of the following types can be used: Integer, Long, Single, Double, Variant.
<i>start</i>	Initial value for <i>counter</i> . The first time through the loop, <i>counter</i> is assigned this value.
<i>end</i>	Final value for <i>counter</i> . The <i>statements</i> will continue executing until <i>counter</i> is equal to <i>end</i> .
<i>increment</i>	<p>Amount added to <i>counter</i> each time through the loop. If <i>end</i> is greater than <i>start</i>, then <i>increment</i> must be positive. If <i>end</i> is less than <i>start</i>, then <i>increment</i> must be negative.</p> <p>If <i>increment</i> is not specified, then 1 is assumed. The expression given as <i>increment</i> is evaluated only once. Changing the step during execution of the loop will have no effect.</p>
<i>statements</i>	<p>Any number of WM Basic statements.</p> <p>The <code>For . . . Next</code> statement continues executing until an <code>Exit For</code> statement is encountered when <i>counter</i> is greater than <i>end</i>.</p> <p><code>For . . . Next</code> statements can be nested. In such a case, the <code>Next [counter]</code> statement applies to the innermost <code>For . . . Next</code>.</p> <p>The <code>Next</code> clause can be optimized for nested next loops by separating each counter with a comma. The ordering of the counters must be consistent with the nesting order (innermost counter appearing before outermost counter). The following example shows two equivalent <code>For</code> statements:</p> <pre> For i = 1 To 10 For j = 1 To 10 Next j Next i </pre> <pre> For i = 1 To 10 For j = 1 To 10 Next j,i </pre> <p>A <code>Next</code> clause appearing by itself (with no <i>counter</i> variable) matches the innermost <code>For</code> loop.</p> <p>The <i>counter</i> variable can be changed within the loop but will have no effect on the number of times the loop will execute.</p>

Example

```

Sub Main()
    'This example constructs a truth table for the OR statement
    'using nested For...Next loops.
    For x = -1 To 0
        For y = -1 To 0
            Z = x Or y
            msg = msg & Format(Abs(x%), "0") & " Or "
            msg = msg & Format(Abs(y%), "0") & " = "
            msg = msg & Format(Z, "True/False") & Basic.Eoln$
        Next y
    Next x
    MsgBox msg
End Sub

```

See Also Do...Loop (statement); While...WEnd (statement).

Platform(s) Windows and Macintosh.

Platform Notes: Due to errors in program logic, you can inadvertently create infinite loops in your code. Under Windows, you can break out of infinite loops using Ctrl+Break.

Platform Notes: Due to errors in program logic, you can inadvertently create infinite loops in your code. On the Macintosh, you can break out of infinite loops using Command+Period.

Format, Format\$ (functions)

Syntax Format[\$](*expression* [, *Userformat\$*])

Description Returns a String formatted to user specification.

Comments `Format$` returns a `String`, whereas `Format` returns a `String` variant.

The `Format$`/`Format` functions take the following parameters:

Parameter	Description
<i>expression</i>	String or numeric expression to be formatted.
<i>Userformat\$</i>	<p>Format expression that can be either one of the built-in WM Basic formats or a user-defined format consisting of characters that specify how the expression should be displayed.</p> <p>String, numeric, and date/time formats cannot be mixed in a single <i>Userformat\$</i> expression.</p> <p>If <i>Userformat\$</i> is omitted and the expression is numeric, then these functions perform the same function as the <code>Str\$</code> or <code>Str</code> statements, except that they do not preserve a leading space for positive values.</p> <p>If <i>expression</i> is <code>Null</code>, then a zero-length string is returned.</p> <p>Built-In Formats</p> <p>To format numeric expressions, you can specify one of the built-in formats. There are two categories of built-in formats: one deals with numeric expressions and the other with date/time values. The following tables list the built-in numeric and date/time format strings, followed by an explanation of what each does.</p>

Numeric Formats

Format	Description
General number	Display the numeric expression as is, with no additional formatting.
Currency	Displays the numeric expression as currency, with thousands separator if necessary.
Fixed	Displays at least one digit to the left of the decimal separator and two digits to the right.
Standard	Displays the numeric expression with thousands separator if necessary. Displays at least one digit to the left of the decimal separator and two digits to the right.
Percent	Displays the numeric expression multiplied by 100. A percent sign (%) will appear at the right of the formatted output. Two digits are displayed to the right of the decimal separator.
Scientific	Displays the number using scientific notation. One digit appears before the decimal separator and two after.
Yes/No	Displays No if the numeric expression is 0. Displays Yes for all other values.

True/False	Displays False if the numeric expression is 0. Displays True for all other values.
On/Off	Displays Off if the numeric expression is 0. Displays On for all other values.

Date/Time Formats

Format	Description
General date	Displays the date and time. If there is no fractional part in the numeric expression, then only the date is displayed. If there is no integral part in the numeric expression, then only the time is displayed. Output is in the following form: 1/1/95 01:00:00 AM.
Long date	Displays a long date.
Medium date	Displays a medium date—prints out only the abbreviated name of the month.
Short date	Displays a short date.
Long time	Displays the long time. The default is: h:mm:ss.
Medium time	Displays the time using a 12-hour clock. Hours and minutes are displayed, and the AM/PM designator is at the end.
Short time	Displays the time using a 24-hour clock. Hours and minutes are displayed.

User-Defined Formats

In addition to the built-in formats, you can specify a user-defined format by using characters that have special meaning when used in a format expression. The following tables list the characters you can use for numeric, string, and date/time formats and explain their functions.

Numeric Formats

Character	Meaning
Empty string	Displays the numeric expression as is, with no additional formatting.
0	This is a digit placeholder. Displays a number or a 0. If a number exists in the numeric expression in the position where the 0 appears, the number will be displayed. Otherwise, a 0 will be displayed. If there are more 0s in the format string than there are digits, the leading and trailing 0s are displayed without modification.
#	This is a digit placeholder. Displays a number or nothing. If a number exists in the numeric expression in the position where the number sign appears, the number will be displayed. Otherwise, nothing will be displayed. Leading and trailing 0s are not displayed.

.	<p>This is the decimal placeholder.</p> <p>Designates the number of digits to the left of the decimal and the number of digits to the right. The character used in the formatted string depends on the decimal placeholder, as specified by your locale.</p>
%	<p>This is the percentage operator.</p> <p>The numeric expression is multiplied by 100, and the percent character is inserted in the same position as it appears in the user-defined format string.</p>
,	<p>This is the thousand separator.</p> <p>The common use for the thousands separator is to separate thousands from hundreds. To specify this use, the thousands separator must be surrounded by digit placeholders. Commas appearing before any digit placeholders are specified are just displayed. Adjacent commas with no digit placeholders specified between them and the decimal mean that the number should be divided by 1,000 for each adjacent comma in the format string. A comma immediately to the left of the decimal has the same function. The actual thousands separator character used depends on the character specified by your locale.</p>
E-E+e-e+	<p>These are the scientific notation operators, which display the number in scientific notation. At least one digit placeholder must exist to the left of E-, E+, e-, or e+. Any digit placeholders displayed to the left of E-, E+, e-, or e+ determine the number of digits displayed in the exponent. Using E+ or e+ places a + in front of positive exponents and a - in front of negative exponents. Using E- or e- places a - in front of negative exponents and nothing in front of positive exponents.</p>
:	<p>This is the time separator.</p> <p>Separates hours, minutes, and seconds when time values are being formatted. The actual character used depends on the character specified by your locale.</p>
/	<p>This is the date separator.</p> <p>Separates months, days, and years when date values are being formatted. The actual character used depends on the character specified by your locale.</p>
-\$ () space	<p>These are the literal characters you can display.</p> <p>To display any other character, you should precede it with a backslash or enclose it in quotes.</p>
\	<p>This designates the next character as a displayed character.</p> <p>To display characters, precede them with a backslash. To display a backslash, use two backslashes. Double quotation marks can also be used to display characters. Numeric formatting characters, date/time formatting characters, and string formatting characters cannot be displayed without a preceding backslash.</p>

"ABC"	Displays the text between the quotation marks, but not the quotation marks. To designate a double quotation mark within a format string, use two adjacent double quotation marks.
*	This will display the next character as the fill character. Any empty space in a field will be filled with the specified fill character. Numeric formats can contain one to three parts. Each part is separated by a semicolon. If you specify one format, it applies to all values. If you specify two formats, the first applies to positive values and the second to negative values. If you specify three formats, the first applies to positive values, the second to negative values, and the third to 0s. If you include semicolons with no format between them, the format for positive values is used.

String Formats

Character	Meaning
@	This is a character placeholder. Displays a character if one exists in the expression in the same position; otherwise, displays a space. Placeholders are filled from right to left unless the format string specifies left to right.
&	This is a character placeholder. Displays a character if one exists in the expression in the same position; otherwise, displays nothing. Placeholders are filled from right to left unless the format string specifies left to right.
<	This character forces lowercase. Displays all characters in the expression in lowercase.
>	This character forces uppercase. Displays all characters in the expression in uppercase.
!	This character forces placeholders to be filled from left to right. The default is right to left.

Date/Time Formats

Character	Meaning
c	Displays the date as dddd and the time as tttt. Only the date is displayed if no fractional part exists in the numeric expression. Only the time is displayed if no integral portion exists in the numeric expression.
d	Displays the day without a leading 0 (1–31).
dd	Displays the day with a leading 0 (01–31).
ddd	Displays the day of the week abbreviated (Sun–Sat).

dddd	Displays the day of the week (Sunday–Saturday).
dddddd	Displays the date as a short date.
ddddddd	Displays the date as a long date.
w	Displays the number of the day of the week (1–7). Sunday is 1; Saturday is 7.
ww	Displays the week of the year (1–53).
m	Displays the month without a leading 0 (1–12). If m immediately follows h or hh, m is treated as minutes (0–59).
mm	Displays the month with a leading 0 (01–12). If mm immediately follows h or hh, mm is treated as minutes with a leading 0 (00–59).
mmm	Displays the month abbreviated (Jan–Dec).
mmmm	Displays the month (January–December).
q	Displays the quarter of the year (1–4).
y	Displays the day of the year (1–366).
yy	Displays the year, not the century (00–99).
yyyy	Displays the year (1000–9999).
h	Displays the hour without a leading 0 (0–24).
hh	Displays the hour with a leading 0 (00–24).
n	Displays the minute without a leading 0 (0–59).
nn	Displays the minute with a leading 0 (00–59).
s	Displays the second without a leading 0 (0–59).
ss	Displays the second with a leading 0 (00–59).
tttttt	Displays the time. A leading 0 is displayed if specified by your locale.
AM/PM	Displays the time using a 12-hour clock. Displays an uppercase AM for time values before 12 noon. Displays an uppercase PM for time values after 12 noon and before 12 midnight.
am/pm	Displays the time using a 12-hour clock. Displays a lowercase am or pm at the end.
A/P	Displays the time using a 12-hour clock. Displays an uppercase A or P at the end.
a/p	Displays the time using a 12-hour clock. Displays a lowercase a or p at the end.

AMPM

Displays the time using a 12-hour clock. Displays the string `s1159` for values before 12 noon and `s2359` for values after 12 noon and before 12 midnight.

Example

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a# = 1199.234
    msg = "Some general formats for '" & a# & "' are:"
    msg = msg & Format$(a#, "General Number") & crlf
    msg = msg & Format$(a#, "Currency") & crlf
    msg = msg & Format$(a#, "Standard") & crlf
    msg = msg & Format$(a#, "Fixed") & crlf
    msg = msg & Format$(a#, "Percent") & crlf
    msg = msg & Format$(a#, "Scientific") & crlf
    msg = msg & Format$(True, "Yes/No") & crlf
    msg = msg & Format$(True, "True/False") & crlf
    msg = msg & Format$(True, "On/Off") & crlf
    msg = msg & Format$(a#, "0,0.00") & crlf
    msg = msg & Format$(a#, "##,###,###.###") & crlf
    MsgBox msg

    da$ = Date$
    msg = "Some date formats for '" & da$ & "' are:"
    msg = msg & Format$(da$, "General Date") & crlf
    msg = msg & Format$(da$, "Long Date") & crlf
    msg = msg & Format$(da$, "Medium Date") & crlf
    msg = msg & Format$(da$, "Short Date") & crlf
    MsgBox msg

    ti$ = Time$
    msg = "Some time formats for '" & ti$ & "' are:"
    msg = msg & Format$(ti$, "Long Time") & crlf
    msg = msg & Format$(ti$, "Medium Time") & crlf
    msg = msg & Format$(ti$, "Short Time") & crlf
    MsgBox msg
End Sub
```

See Also `Str`, `Str$` (functions); `CStr` (function).

Platform(s) Windows and Macintosh.

Platform Under Windows, default date/time formats are read from the [Intl]

Notes: section of the win.ini file.

Windows

FreeFile (function)

Syntax `FreeFile[()]`

Description Returns an Integer containing the next available file number.

Comments The number returned is suitable for use in the `Open` statement and will always be between 1 and 255 inclusive.

Example 'This example assigns A to the next free file number and displays it
'in a dialog box.

```
Sub Main()  
    a = FreeFile  
    MsgBox "The next free file number is: " & a  
End Sub
```

See Also FileAttr (function); Open (statement).

Platform(s) Windows and Macintosh.

Function...End Function (statement)

Syntax [Private | Public] [Static] Function *name*[(*arglist*)] [As *ReturnType*]
 [*statements*]
End Sub

where *arglist* is a comma-separated list of the following (up to 30 arguments are allowed):

[Optional] [ByVal | ByRef] *parameter* [()] [As *type*]

Description Creates a user-defined function.

Comments The <code>Function</code> statement has the following parts:	
Part	Description
<code>Private</code>	Indicates that the function being defined cannot be called from other scripts.
<code>Public</code>	Indicates that the function being defined can be called from other scripts. If both the <code>Private</code> and <code>Public</code> keywords are missing, then <code>Public</code> is assumed.
<code>Static</code>	Recognized by the compiler but currently has no effect.
<i>name</i>	<p>Name of the function, which must follow WM Basic naming conventions:</p> <ol style="list-style-type: none"> 1. Must start with a letter. 2. May contain letters, digits, and the underscore character (<code>_</code>). Punctuation and type-declaration characters are not allowed. The exclamation point (<code>!</code>) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character. 3. Must not exceed 80 characters in length. <p>Additionally, the <i>name</i> parameter can end with an optional type-declaration character specifying the type of data returned by the function (i.e., any of the following characters: <code>%</code>, <code>&</code>, <code>!</code>, <code>#</code>, <code>@</code>).</p>
<code>Optional</code>	<p>Keyword indicating that the parameter is optional. All optional parameters must be of type <code>Variant</code>. Furthermore, all parameters that follow the first optional parameter must also be optional.</p> <p>If this keyword is omitted, then the parameter is required.</p> <hr/> <p>Note: You can use the <code>IsMissing</code> function to determine if an optional parameter was actually passed by the caller.</p> <hr/>
<code>ByVal</code>	Keyword indicating that <i>parameter</i> is passed by value.

<code>ByRef</code>	Keyword indicating that <i>parameter</i> is passed by reference. If neither the <code>ByVal</code> nor the <code>ByRef</code> keyword is given, then <code>ByRef</code> is assumed.
<i>parameter</i>	Name of the parameter, which must follow the same naming conventions as those used by variables. This name can include a type-declaration character, appearing in place of <code>As type</code> .
<i>type</i>	Type of the parameter (i.e., <code>Integer</code> , <code>String</code> , and so on). Arrays are indicated with parentheses. For example, an array of integers would be declared as follows:

```
Function Test(a() As Integer)
End Function
```

<i>ReturnType</i>	Type of data returned by the function. If the return type is not given, then <code>Variant</code> is assumed. The <i>ReturnType</i> can only be specified if the function name (i.e., the <i>name</i> parameter) does not contain an explicit type-declaration character.
-------------------	---

A function returns to the caller when either of the following statements is encountered:

```
End Function
Exit Function
```

Functions can be recursive.

Returning Values from Functions

To assign a return value, an expression must be assigned to the name of the function, as shown below:

```
Function TimesTwo(a As Integer) As Integer
    TimesTwo = a * 2
End Function
```

If no assignment is encountered before the function exits, then one of the following values is returned:

Value	Data Type Returned by the Function
0	<code>Integer</code> , <code>Long</code> , <code>Single</code> , <code>Double</code> , <code>Currency</code>
Zero-length string	<code>String</code>
Nothing	<code>Object</code> (or any data object)
Empty	<code>Variant</code>
December 30, 1899	<code>Date</code>
False	<code>Boolean</code>

The type of the return value is determined by the *As ReturnType* clause on the `Function` statement itself. As an alternative, a type-declaration character can be added to the `Function` name. For example, the following two definitions of `Test` both return `String` values:

```
Function Test() As String
    Test = "Hello, world"
End Function

Function Test$()
    Test = "Hello, world"
End Function
```

Passing Parameters to Functions

Parameters are passed to a function either by value or by reference, depending on the declaration of that parameter in *arglist*. If the parameter is declared using the `ByRef` keyword, then any modifications to that passed parameter within the function change the value of that variable in the caller. If the parameter is declared using the `ByVal` keyword, then the value of that variable cannot be changed in the called function. If neither the `ByRef` or `ByVal` keywords are specified, then the parameter is passed by reference.

You can override passing a parameter by reference by enclosing that parameter within parentheses. For instance, the following example passes the variable `j` by reference, regardless of how the third parameter is declared in the *arglist* of `UserFunction`:

```
i = UserFunction(10,12,(j))
```

Optional Parameters

WM Basic allows you to skip parameters when calling functions, as shown in the following example:

```
Function Test(a%,b%,c%) As Variant
End Function

Sub Main
  a = Test(1,,4)      'Parameter 2 was skipped.
End Sub
```

You can skip any parameter with the following restrictions:

1. The call cannot end with a comma. For instance, using the above example, the following is not valid:

```
a = Test(1,,)
```
2. The call must contain the minimum number of parameters as required by the called function. For instance, using the above example, the following are invalid:

```
a = Test(,1) 'Only passes two out of three required parameters.
a = Test(1,2)'Only passes two out of three required parameters.
```

When you skip a parameter in this manner, WM Basic creates a temporary variable and passes this variable instead. The value of this temporary variable depends on the data type of the corresponding parameter in the argument list of the called function, as described in the following table:

Value	Data Type
0	Integer, Long, Single, Double, Currency
Zero-length string	String
Nothing	Object (or any data object)
Error	Variant
December 30, 1899	Date
False	Boolean

Within the called function, you will be unable to determine if a parameter was skipped unless the parameter was declared as a variant in the argument list of the function. In this case, you can use the `IsMissing` function to determine if the parameter was skipped:

```
Function Test(a,b,c)
  If IsMissing(a) Or IsMissing(b) Then Exit Sub
End Function
```

Example **Function** Factorial(n%) As Integer
 'This function calculates N! (N-factorial).
 f% = 1
 For i = n To 2 Step -1
 f = f * i
 Next i
 Factorial = f
End Function

Sub Main()
 'This example calls user-defined function Factorial and displays the
 'result in a dialog box.
 a% = 0
 Do While a% < 2
 a% = Val(InputBox\$("Enter an integer number greater than
 2.", "Compute Factorial"))
 Loop
 b# = Factorial(a%)
 MsgBox "The factorial of " & a% & " is: " & b#
 End Sub

See Also Sub...End Sub (statement)

Platform(s) Windows and Macintosh.

Fv (function)

Syntax Fv(*Rate, Nper, Pmt, Pv, Due*)

Description Calculates the future value of an annuity based on periodic fixed payments and a constant rate of interest.

Comments An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The Fv function requires the following parameters:

Parameter	Description
<i>Rate</i>	Double representing the interest rate per period. Make sure that annual rates are normalized for monthly periods (divided by 12).
<i>NPer</i>	Double representing the total number of payments (periods) in the annuity.
<i>Pmt</i>	Double representing the amount of each payment per period. Payments are entered as negative values, whereas receipts are entered as positive values.
<i>Pv</i>	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, whereas in the case of a retirement annuity, the present value would be the amount of the fund.
<i>Due</i>	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period.

Rate and *NPer* values must be expressed in the same units. If *Rate* is expressed as a percentage per month, then *NPer* must also be expressed in months. If *Rate* is an annual rate, then the *NPer* must also be given in years.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example 'This example calculates the future value of 100 dollars paid periodically for a period of 10 years (120 months) at a rate of 10% per year (or .10/12 per month) with payments made on the first of the month. The value is displayed in a dialog box. Note that payments are negative values.

```
Sub Main()  
    a# = Fv((.10/12),120,-100.00,0,1)  
    MsgBox "Future value is: " & Format(a#,"Currency")  
End Sub
```

See Also IRR (function); MIRR (function); Npv (function); Pv (function).

Platform(s) Windows and Macintosh.

Get (statement)

Syntax Get [#] *filename*, [*recordnumber*], *variable*

Description Retrieves data from a random or binary file and stores that data into the specified variable.

Comments The `Get` statement accepts the following parameters:

Parameter	Description
<i>filenumber</i>	Integer used by WM Basic to identify the file. This is the same number passed to the <code>Open</code> statement.
<i>recordnumber</i>	<p>Long specifying which record is to be read from the file.</p> <p>For <code>binary</code> files, this number represents the first byte to be read starting with the beginning of the file (the first byte is 1). For <code>random</code> files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647.</p> <p>If the <i>recordnumber</i> parameter is omitted, the next record is read from the file (if no records have been read yet, then the first record in the file is read). When this parameter is omitted, the commas must still appear, as in the following example:</p> <pre>Get #1,,recvar</pre> <p>If <i>recordnumber</i> is specified, it overrides any previous change in file position specified with the <code>Seek</code> statement.</p>
<i>variable</i>	<p>Variable into which data will be read. The type of the variable determines how the data is read from the file, as described below.</p> <p>With random files, a runtime error will occur if the length of the data being read exceeds the <i>reclen</i> parameter specified with the <code>Open</code> statement. If the length of the data being read is less than the record length, the file pointer is advanced to the start of the next record. With binary files, the data elements being read are contiguous the file pointer is never advanced.</p>

Variable Types

The type of the *variable* parameter determines how data will be read from the file. It can be any of the following types:

Variable Type	File Storage Description
Integer	2 bytes are read from the file.
Long	4 bytes are read from the file.

String (variable-length) In binary files, variable-length strings are read by first determining the specified string variable's length and then reading that many bytes from the file. For example, to read a string of eight characters:

```
s$ = String$(8," ")
Get #1,,s$
```

In random files, variable-length strings are read by first reading a 2-byte length and then reading that many characters from the file.

String (fixed-length) Fixed-length strings are read by reading a fixed number of characters from the file equal to the string's declared length.

Double 8 bytes are read from the file (IEEE format).

Single 4 bytes are read from the file (IEEE format).

Date 8 bytes are read from the file (IEEE double format).

Boolean 2 bytes are read from the file. Nonzero values are `True`, and zero values are `False`.

Variant A 2-byte `VarType` is read from the file, which determines the format of the data that follows. Once the `VarType` is known, the data is read individually, as described above. With user-defined errors, after the 2-byte `VarType`, a 2-byte unsigned integer is read and assigned as the value of the user-defined error, followed by 2 additional bytes of information about the error.

The exception is with strings, which are always preceded by a 2-byte string length.

User-defined types Each member of a user-defined data type is read individually.

In binary files, variable-length strings within user-defined types are read by first reading a 2-byte length followed by the string's content. This storage is different from variable-length strings outside of user-defined types.

When reading user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.

Arrays Arrays cannot be read from a file using the `Get` statement.

Objects

Object variables cannot be read from a file using the Get statement.

Example 'This example opens a file for random write, then writes ten records into the file with the values 10...50. Then the file is closed and reopened in random mode for read, and the records are read with the Get statement. The result is displayed in a message box.

```
Sub Main()
    Open "test.dat" For Random Access Write As #1
    For x = 1 to 10
        y% = x * 10
        Put #1,x,y
    Next x
    Close

    Open "test.dat" For Random Access Read As #1
    For y = 1 to 5
        Get #1,y,x%
        msg = msg & "Record " & y & ": " & x% & Basic.Eoln$
    Next y

    MsgBox msg
    Close
End Sub
```

See Also Open (statement); Put (statement); Input# (statement); Line Input# (statement); Input, Input\$ (functions).

Platform(s) Windows and Macintosh.

GetAttr (function)

Syntax GetAttr(filename\$)

Description Returns an Integer containing the attributes of the specified file.

Comments The attribute value returned is the sum of the attributes set for the file. The value of each attribute is as follows:

Constant	Value	Includes
ebNormal	0	Read-only files, archive files, subdirectories,
ebReadOnly	1	and files with no attributes. Read-only files
ebHidden	2	Hidden files
ebSystem	4	System files
ebVolume	8	Volume label
ebDirectory	16	DOS subdirectories
ebArchive	32	Files that have changed since the last backup
ebNone	64	Files with no attributes

To determine whether a particular attribute is set, you can And the values shown above with the value returned by GetAttr. If the result is True, the attribute is set, as shown below:

```
Dim w As Integer
w = GetAttr("sample.txt")
If w And ebReadOnly Then MsgBox "This file is read-only."
```

Example 'This example tests to see whether the file test.dat exists.
'If it does not, then it creates the file. The file attributes are
'then retrieved with the GetAttr function, and the result is
'displayed.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    If Not FileExists("test.dat") Then
        Open "test.dat" For Random Access Write As #1
        Close
    End If

    y% = GetAttr("test.dat")
    If y% And ebNone Then msg = msg & "No archive bit is set." & crlf
    If y% And ebReadOnly Then msg = msg & "The read-only bit is set." & crlf
    If y% And ebHidden Then msg = msg & "The hidden bit is set." & crlf
    If y% And ebSystem Then msg = msg & "The system bit is set." & crlf
    If y% And ebVolume Then msg = msg & "The volume bit is set." & crlf
    If y% And ebDirectory Then msg = msg & "The directory bit is set." & crlf
    If y% And ebArchive Then msg = msg & "The archive bit is set."

    MsgBox msg
    Kill "test.dat"
End Sub
```


See Also SetAttr (statement); FileAttr (function).

Platform(s) Windows and Macintosh.

Platform Under Windows, these attributes are the same as those used by DOS.

Notes:
Windows

GetCheckBox (function)

Syntax GetCheckBox(*name\$* | *id*)

Description Returns an Integer representing the state of the specified check box.

Comments This function is used to determine the state of a check box, given its name or ID. The returned value will be one of the following:

Returned Value	Description
0	Check box contains no check.
1	Check box contains a check.
2	Check box is grayed.

The GetCheckBox function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the check box.
<i>id</i>	Integer specifying the ID of the check box.

Note: The GetCheckBox function is used to retrieve the state of a check box in another application's dialog box. Use the DlgValue function to retrieve the state of a check box in a dynamic dialog box.

Example 'This example toggles the Match Case check box in the Find dialog box.

```
Sub Main()
    Menu "Search.Find"
    If GetCheckBox("Match Case") = 0 Then
        SetCheckBox "Match Case", 1
    Else
        SetCheckBox "Match Case", 0
    End If
End Sub
```

See Also CheckBoxExists (function); CheckBoxEnabled (function); SetCheckBox (statement); DlgValue (function).

Platform(s) Windows.

GetComboBoxItem\$ (function)

Syntax `GetComboBoxItem$(name$ | id [,ItemNumber])`

Description Returns a String containing the text of an item within a combo box.

Comments The GetComboBoxItem\$ function takes the following parameters:

Parameter	Description
<i>name\$</i>	String specifying the name of the combo box containing the item to be returned. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the combo box containing the item to be returned.
<i>ItemNumber</i>	Integer containing the line number of the desired combo box item to be returned. If omitted, then the currently selected item in the combo box is returned. The combo box must exist within the current window or dialog box; otherwise, a runtime error is generated. A zero-length string will be returned if the combo box does not contain textual items.
Note: The GetComboBoxItem\$ function is used to retrieve the current item of a combo box in another application's dialog box. Use the DlgText function to retrieve the current item of a combo box in a dynamic dialog box.	

Example 'This example retrieves the last item from a combo box.

```
Sub Main()
    last% = GetComboBoxItemCount("Directories:")
    s$ = GetComboBoxItem$("Directories:",last% - 1) 'Number is 0-
    based
    MsgBox "The last item in the combo box is " & s$
End Sub
```

See Also ComboBoxEnabled (function); ComboBoxExists (function);
GetComboBoxItemCount (function); SelectComboBoxItem (statement).

Platform(s) Windows.

GetComboBoxItemCount (function)

Syntax `GetComboBoxItemCount(name$ | id)`

Description Returns an Integer containing the number of items in the specified combo box.

Comments The GetComboBoxItemCount function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the combo box. The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the combo box. A runtime error is generated if the specified combo box does not exist within the current window or dialog box.
Note: The GetComboBoxItemCount function is used to determine the number of items in a combo box in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example 'This example copies all the items out of a combo box and into an array.

```
Sub Main()
    Dim MyList$()
    last% = GetComboBoxItemCount("Directories:")
    ReDim MyList$(0 To last - 1)
    For i = 0 To last - 1
        MyList$(i) = GetComboBoxItem$("Directories:", i)
    Next i
End Sub
```

See Also ComboBoxEnabled (function); ComboBoxExists (function); GetComboBoxItem\$ (function); SelectComboBoxItem (statement).

Platform(s) Windows.

GetEditText\$ (function)

Syntax GetEditText\$(*name\$* | *id*)

Description Returns a String containing the content of the specified text box control.

Comments The `GetEditText$` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the text box whose content will be returned. The name of a text box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a text box. A runtime error is generated if a text box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the text box whose content will be returned. A runtime error is generated if a text box control with the given name or ID cannot be found within the active window.
Note: The <code>GetEditText\$</code> function is used to retrieve the content of a text box in another application's dialog box. Use the <code>DlgText\$</code> function to retrieve the content of a text box in a dynamic dialog box.	

Example 'This example retrieves the filename and prepends it with the current 'directory.

```
Sub Main()  
    s$ = GetEditText$("Filename:")           'Retrieve the content of  
the edit control.  
    s$ = CurDir$ & Basic.PathSeparator & s$ 'Prepend the current  
directory.  
    SetEditText "Filename:",s$               'Put it back.  
End Sub
```

See Also `EditEnabled` (function); `EditExists` (function); `SetEditText` (statement).

Platform(s) Windows.

GetListBoxItem\$ (function)

Syntax `GetListBoxItem$(name$ | id,[item])`

Description Returns a `String` containing the specified item in a list box.

Comments The `GetListBoxItem$` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String specifying the name of the list box containing the item to be returned. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the list box containing the item to be returned.
<i>item</i>	Integer containing the line number of the desired list box item to be returned. This number must be between 1 and the number of items in the list box. If omitted, then the currently selected item in the list box is returned. A runtime error is generated if the specified list box cannot be found within the active window.
Note: The <code>GetListBoxItem\$</code> function is used to retrieve an item from a list box in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example 'This example sees whether my name appears as an item in the "Users" list box.

```
Sub Main()
    last% = GetListBoxItemCount("Users")
    IsThere = False
    For i = 0 To last% - 1      'Number is zero-based.
        If GetListBoxItem$("Users",i) = Net.User$ Then IsThere = True
    Next i
    If IsThere Then MsgBox "I am a member!",vbOKOnly
End Sub
```

See Also `GetListBoxItemCount (function)`; `ListBoxEnabled (function)`; `ListBoxExists (function)`; `SelectListBoxItem (statement)`.

Platform(s) Windows.

GetListBoxItemCount (function)

Syntax `GetListBoxItemCount(name$ | id)`

Description Returns an Integer containing the number of items in a specified list box.

Comments The `GetListBoxItemCount` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the list box. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the list box. A runtime error is generated if the specified list box cannot be found within the active window.
Note: The <code>GetListBoxItemCount</code> function is used to retrieve the number of items in a list box in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example See `GetListBoxItem$` (function).

See Also `GetListBoxItem$` (function); `ListBoxEnabled` (function); `ListBoxExists` (function); `SelectListBoxItem` (statement).

Platform(s) Windows.

GetObject (function)

Syntax `GetObject(filename$ [,class$])`

Description Returns the object specified by *filename\$* or returns a previously instantiated object of the given *class\$*.

Comments This function is used to retrieve an existing OLE automation object, either one that comes from a file or one that has previously been instantiated.

The *filename\$* argument specifies the full pathname of the file containing the object to be activated. The application associated with the file is determined by OLE at runtime. For example, suppose that a file called `c:\docs\resume.doc` was created by a word processor called `wordproc.exe`. The following statement would invoke `wordproc.exe`, load the file called `c:\docs\resume.doc`, and assign that object to a variable:

```
Dim doc As Object
Set doc = GetObject("c:\docs\resume.doc")
```

To activate a part of an object, add an exclamation point to the filename followed by a string representing the part of the object that you want to activate. For example, to activate the first three pages of the document in the previous example:

```
Dim doc As Object
Set doc = GetObject("c:\docs\resume.doc!P1-P3")
```

The `GetObject` function behaves differently depending on whether the first parameter is omitted. The following table summarizes the different behaviors of `GetObject`:

Filename\$	Class\$	GetObject Returns
Omitted	Specified	Reference to an existing instance of the specified object. A runtime error results if the object is not already loaded.
" "	Specified	Reference to a new object (as specified by <i>class\$</i>). A runtime error occurs if an object of the specified class cannot be found. This is the same as <code>CreateObject</code> .
Specified	Omitted	Default object from <i>filename\$</i> . The application to activate is determined by OLE based on the given filename.
Specified	Specified	Object given by <i>class\$</i> from the file given by <i>filename\$</i> . A runtime error occurs if an object of the given class cannot be found in the given file.

Examples 'This first example instantiates the existing copy of Excel.

```
Dim Excel As Object
Set Excel = GetObject(,"Excel.Application")
```

'This second example loads the OLE server associated with a document.

```
Dim MyObject As Object
Set MyObject = GetObject("c:\documents\resume.doc",)
```

See Also CreateObject (function); Object (data type).

Platform(s) Windows and Macintosh.

GetOption (function)

Syntax GetOption(*name\$* | *id*)

Description Returns True if the option is set; returns False otherwise.

Comments The GetOption function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the option button.
<i>id</i>	Integer containing the ID of the option button. The <i>id</i> must be used when the name of the option button is not known in advance. The option button must exist within the current window or dialog box. A runtime error will be generated if the specified option button does not exist.

Note: The GetOption function is used to retrieve the state of an option button in another application's dialog box. Use the DlgValue function to retrieve the state of an option button in a dynamic dialog box.

Example 'This example figures out which option is set in the Desktop dialog box of the Control Panel.

```
Sub Main()
    id = Shell("control",7)           'Run the Control Panel.
    WinActivate "Control Panel"       'Activate the Control Panel
window.
    Menu "Settings.Desktop"          'Select Desktop dialog box.
    WinActivate "Control Panel|Desktop" 'Activate it.
    If GetOption("Tile") Then        'Retrieve which option is
set.
        MsgBox "Your wallpaper is tiled." 'The Tile option is
currently set.
    Else
        MsgBox "Your wallpaper is centered." 'The Centered option is
currently set.
    End If
End Sub
```


See Also `OptionEnabled` (function); `OptionExists` (function); `SetOption` (statement).

Platform(s) Windows.

Global (statement)

See `Global` (statement).

Platform(s) Windows and Macintosh.

GoSub (statement)

Syntax `GoSub label`

Description Causes execution to continue at the specified label.

Comments Execution can later be returned to the statement following the `GoSub` by using the `Return` statement.

The *label* parameter must be a label within the current function or subroutine. `GoSub` outside the context of the current function or subroutine is not allowed.

Example 'This example gets a name from the user and then branches to a 'subroutine to check the input. If the user clicks Cancel or enters a 'blank name, the program terminates; otherwise, the name is set to 'MICHAEL, and a message is displayed.

```
Sub Main()  
    uname$ = Ucase$(InputBox$("Enter your name:", "Enter Name"))  
    GoSub CheckName  
    MsgBox "Hello, " & uname$  
    Exit Sub  
  
CheckName:  
    If (uname$ = "") Then  
        GoSub BlankName  
    ElseIf uname$ = "MICHAEL" Then  
        GoSub RightName  
    Else  
        GoSub OtherName  
    End If  
    Return  
  
BlankName:  
    MsgBox "No name? Clicked Cancel? I'm shutting down."  
    Exit Sub  
  
RightName:  
    Return  
  
OtherName:  
    MsgBox "I am renaming you MICHAEL!"  
    uname$ = "MICHAEL"  
    Return  
End Sub
```

See Also Goto (statement); Return (statement).

Platform(s) Windows and Macintosh.

Goto (statement)

Syntax Goto *label*

Description Transfers execution to the line containing the specified label.

Comments The compiler will produce an error if *label* does not exist.

The *label* must appear within the same subroutine or function as the Goto.

Labels are identifiers that follow these rules:

1. Must begin with a letter.
2. May contain letters, digits, and the underscore character.
3. Must not exceed 80 characters in length.
4. Must be followed by a colon (:).

Labels are not case-sensitive.

Example 'This example gets a name from the user and then branches to a 'statement, depending on the input name. If the name is not MICHAEL, 'it is reset to MICHAEL unless it is null or the user clicks Cancel-- 'in which case, the program displays a message and terminates.

```
Sub Main()
    unname$ = Ucase$(InputBox$("Enter your name:", "Enter Name"))
    If unname$ = "MICHAEL" Then
        Goto RightName
    Else
        Goto WrongName
    End If

WrongName:
    If (unname$ = "") Then
        MsgBox "No name? Clicked Cancel? I'm shutting down."
    Else
        MsgBox "I am renaming you MICHAEL!"
        unname$ = "MICHAEL"
        Goto RightName
    End If
Exit Sub

RightName:
    MsgBox "Hello, MICHAEL!"
End Sub
```

See Also GoSub (statement); Call (statement).

Platform(s) Windows and Macintosh.

Platform To break out of an infinite loop, press Ctrl+Break.

Notes:

Windows

Platform To break out of an infinite loop, press Ctrl+Period.

Notes:

Macintosh

GroupBox (statement)

Syntax `GroupBox X,Y,width,height,title$ [, .Identifier]`

Description Defines a group box within a dialog box template.

Comments This statement can only appear within a dialog box template (i.e., between the `Begin Dialog` and `End Dialog` statements).

The group box control is used for static display only the user cannot interact with a group box control.

Separator lines can be created using group box controls. This is accomplished by creating a group box that is wider than the width of the dialog box and extends below the bottom of the dialog box i.e., three sides of the group box are not visible.

If *title\$* is a zero-length string, then the group box is drawn as a solid rectangle with no title.

The `GroupBox` statement requires the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>title\$</i>	String containing the label of the group box. If <i>title\$</i> is a zero-length string, then no title will appear.
<i>.Identifier</i>	Optional parameter that specifies the name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>). If omitted, then the first two words of <i>title\$</i> are used.

Example 'This example shows the `GroupBox` statement being used both for grouping and as a separator line.

```
Sub Main()
    Begin Dialog OptionsTemplate 16,32,128,84,"Options"
        GroupBox 4,4,116,40,"Window Options"
        CheckBox 12,16,60,8,"Show &Toolbar",.ShowToolbar
        CheckBox 12,28,68,8,"Show &Status Bar",.ShowStatusBar
        GroupBox -12,52,152,48," ",.SeparatorLine
        OKButton 16,64,40,14,.OK
        CancelButton 68,64,40,14,.Cancel
    End Dialog
    Dim OptionsDialog As OptionsTemplate
    Dialog OptionsDialog
End Sub
```

See Also CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Platform(s) Windows and Macintosh.

Hex, Hex\$ (functions)

Syntax Hex[\$](*number*)

Description Returns a String containing the hexadecimal equivalent of *number*.

Comments Hex\$ returns a String, whereas Hex returns a String variant.

The returned string contains only the number of hexadecimal digits necessary to represent the number, up to a maximum of eight.

The *number* parameter can be any type but is rounded to the nearest whole number before converting to hex. If the passed number is an integer, then a maximum of four digits are returned; otherwise, up to eight digits can be returned.

The *number* parameter can be any expression convertible to a number. If *number* is Null, then Null is returned. Empty is treated as 0.

Example 'This example inputs a number and displays it in decimal and 'hex until the input number is 0 or an invalid input.

```
Sub Main()
    Do
        xs$ = InputBox$("Enter a number to convert:", "Hex Convert")
        x = Val(xs$)
        If x <> 0 Then
            MsgBox "Dec: " & x & "    Hex: " & Hex$(x)
        Else
            MsgBox "Goodbye."
        End If
    Loop While x <> 0
End Sub
```

See Also Oct, Oct\$ (functions).

Platform(s) Windows and Macintosh.

HLine (statement)

Syntax HLine [*lines*]

Description Scrolls the window with the focus left or right by the specified number of lines.

Comments The *lines* parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled right by one line.

Example 'This example scrolls the Notepad window to the left by three
'"amounts." Each "amount" is equivalent to clicking the right arrow
'of the horizontal scroll bar once.

```
Sub Main()  
    AppActivate "Notepad"  
    HLine 3      'Move 3 lines in.  
End Sub
```

See Also HPage (statement); HScroll (statement).

Platform(s) Windows.

Hour (function)

Syntax Hour (*time*)

Description Returns the hour of the day encoded in the specified *time* parameter.

Comments The value returned is as an Integer between 0 and 23 inclusive.

The *time* parameter is any expression that converts to a Date.

Example 'This example takes the current time; extracts the hour,
'minute, and second; and displays them as the current time.

```
Sub Main()  
    xt# = TimeValue(Time$())  
    xh# = Hour(xt#)  
    xm# = Minute(xt#)  
    xs# = Second(xt#)  
    MsgBox "The current time is: " & xh# & ":" & xm# & ":" & xs#  
End Sub
```

See Also Day (function); Minute (function); Second (function); Month (function); Year (function); Weekday (function); DatePart (function).

Platform(s) Windows and Macintosh.

HPage (statement)

Syntax HPage [*pages*]

Description Scrolls the window with the focus left or right by the specified number of pages.

Comments The *pages* parameter is an Integer specifying the number of pages to scroll. If this parameter is omitted, then the window is scrolled right by one page.

Example 'This example scrolls the Notepad window to the left by three "amounts." Each "amount" is equivalent to clicking within the horizontal scroll bar on the right side of the thumb mark.

```
Sub Main()
    AppActivate "Notepad"
    HPage 3      'Move 3 pages down.
End Sub
```

See Also HLine (statement); HScroll (statement).

Platform(s) Windows.

HScroll (statement)

Syntax HScroll *percentage*

Description Sets the thumb mark on the horizontal scroll bar attached to the current window.

Comments The position is given as a percentage of the total range associated with that scroll bar. For example, if the *percentage* parameter is 50, then the thumb mark is positioned in the middle of the scroll bar.

Example 'This example centers the thumb mark on the horizontal scroll bar of the Notepad window.

```
Sub Main()
    AppActivate "Notepad"
    HScroll 50      'Jump to the middle of the document.
End Sub
```

See Also HLine (statement); HPage (statement).

Platform(s) Windows.

HWND (object)

Syntax Dim *name* As HWND

Description A data type used to hold window objects.

Comments This data type is used to hold references to physical windows in the operating environment. The following commands operate on HWND objects:

WinActivate	WinClose	WinFind	WinList
WinMaximize	WinMinimize	WinMove	WinRestore
WinSize			

The above language elements support both string and HWND window specifications.

Example 'This example activates the "Main" MDI window within Program Manager.

```
Sub Main()  
    Dim ProgramManager As HWND  
    Dim ProgramManagerMain As HWND  
    Set ProgramManager = WinFind("Program Manager")  
    If ProgramManager Is Not Nothing Then  
        WinActivate ProgramManager  
        WinMaximize ProgramManager  
        Set ProgramManagerMain = WinFind("Program Manager|Main")  
        If ProgramManagerMain Is Not Nothing Then  
            WinActivate ProgramManagerMain  
            WinRestore ProgramManagerMain  
        Else  
            MsgBox "Your Program Manager doesn't have a Main group."  
        End If  
    Else  
        MsgBox "Program Manager is not running."  
    End If  
End Sub
```

See Also **HWND.Value** (property); **WinFind** (function); **WinActivate** (function).

Platform(s) Windows.

HWND.Value (property)

Syntax *window.Value*

Description The default property of an **HWND** object that returns a **Variant** containing a **HANDLE** to the physical window of an **HWND** object variable.

Comments The **Value** property is used to retrieve the operating environment-specific value of a given **HWND** object. The size of this value depends on the operating environment in which the script is executing and thus should always be placed into a **Variant** variable.

This property is read-only.

Example 'This example displays a dialog box containing the class name of 'Program Manager's Main window. It does so using the **.Value** property, 'passing it directly to a Windows' external routine.

```
Declare Sub GetClassName Lib "user" (ByVal Win%,ByVal ClsName$,ByVal  
ClsNameLen%)  
  
Sub Main()  
    Dim ProgramManager As HWND  
    Set ProgramManager = WinFind("Program Manager")  
    ClassName$ = Space(40)  
    GetClassName ProgramManager.Value,ClassName$,Len(ClassName$)  
    MsgBox "The program classname is: " & ClassName$  
End Sub
```


See Also HWND (data type).

Platform(s) Windows.

Platform Under Windows, this value is an Integer.

Notes:
Windows

If...Then...Else (statement)

Syntax 1 If *condition* Then *statements* [Else *else_statements*]

Syntax 2 If *condition* Then
 [*statements*]
[ElseIf *else_condition* Then
 [*elseif_statements*]]
[Else
 [*else_statements*]]
End If

Description Conditionally executes a statement or group of statements.

Comments The single-line conditional statement (syntax 1) has the following parameters:

Parameter	Description
<i>condition</i>	Any expression evaluating to a Boolean value.
<i>statements</i>	One or more statements separated with colons. This group of statements is executed when <i>condition</i> is True.
<i>else_statements</i>	One or more statements separated with colons. This group of statements is executed when <i>condition</i> is False.

278 Working Model Basic User's Manual

The multiline conditional statement (syntax 2) has the following parameters:

Parameter	Description
<i>condition</i>	Any expression evaluating to a Boolean value.
<i>statements</i>	One or more statements to be executed when <i>condition</i> is True.
<i>else_condition</i>	Any expression evaluating to a Boolean value. The <i>else_condition</i> is evaluated if <i>condition</i> is False.
<i>elseif_statements</i>	One or more statements to be executed when <i>condition</i> is False and <i>else_condition</i> is True.
<i>else_statements</i>	One or more statements to be executed when both <i>condition</i> and <i>else_condition</i> are False.

There can be as many ElseIf conditions as required.

Example

'This example inputs a name from the user and checks to see whether it 'is MICHAEL or MIKE using three forms of the If...Then...Else 'statement. It then branches to a statement that displays a welcome 'message depending on the user's name.

```
Sub Main()  
    unname$ = UCase$(InputBox$("Enter your name:", "Enter Name"))  
    If unname$ = "MICHAEL" Then GoSub MikeName  
  
    If unname$ = "MIKE" Then  
        GoSub MikeName  
        Exit Sub  
    End If  
  
    If unname$ = "" Then  
        MsgBox "Since you don't have a name, I'll call you MIKE!"  
        unname$ = "MIKE"  
        GoSub MikeName  
    ElseIf unname$ = "MICHAEL" Then  
        GoSub MikeName  
    Else  
        GoSub OtherName  
    End If  
    Exit Sub  
  
MikeName:  
    MsgBox "Hello, MICHAEL!"  
    Return  
  
OtherName:  
    MsgBox "Hello, " & unname$ & "!"  
    Return  
End Sub
```

See Also Choose (function); Switch (function); IIf (function); Select...Case (statement).

Platform(s) Windows and Macintosh.

If (function)

Syntax `IIf (condition , TrueExpression , FalseExpression)`

Description Returns *TrueExpression* if *condition* is True; otherwise, returns *FalseExpression*.

Comments Both expressions are calculated before `IIf` returns.

The `IIf` function is shorthand for the following construct:

```
If condition Then
    variable = TrueExpression
Else
    variable = FalseExpression
End If
```

Example

```
Sub Main()
    s$ = "Car"
    MsgBox IIf(s$ = "Car", "Nice Car", "Nice Automobile")
End Sub
```

See Also Choose (function); Switch (function); If...Then...Else (statement); Select...Case (statement).

Platform(s) Windows and Macintosh.

Imp (operator)

Syntax `expression1 Imp expression2`

Description Performs a logical or binary implication on two expressions.

Comments If both expressions are either Boolean, Boolean variants, or Null variants, then a logical implication is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

Binary Implication

If the two expressions are Integer, then a binary implication is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary implication is then performed, returning a Long result.

Binary implication forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions, according to the following table:

1	Imp1	=	1	Example:
0	Imp1	=	1	5 01101001
1	Imp0	=	0	6 10101010
0	Imp0	=	1	Imp10111110

Example 'This example compares the result of two expressions to determine whether one implies the other.

```
Sub Main()
    a = 10 : b = 20 : c = 30 : d = 40

    If (a < b) Imp (c < d) Then
        MsgBox "a is less than b implies that c is less than d."
    Else
        MsgBox "a is less than b does not imply that c is less than d."
    End If

    If (a < b) Imp (c > d) Then
        MsgBox "a is less than b implies that c is greater than d."
    Else
        MsgBox "a is less than b does not imply that c is greater than d."
    End If
End Sub
```

See Also Operator Precedence (topic); Or (operator); Xor (operator); Eqv (operator); And (operator).

Platform(s) Windows and Macintosh.

Inline (statement)

Syntax `Inline name [parameters]
anytext
End Inline`

Description Allows execution or interpretation of a block of text.

Comments The `Inline` statement takes the following parameters:

Parameter	Description
-----------	-------------

<i>name</i>	Identifier specifying the type of inline statement.
-------------	---

<i>parameters</i>	Comma-separated list of parameters.
-------------------	-------------------------------------

<i>anytext</i>	Text to be executed by the <code>Inline</code> statement. This text must be in a format appropriate for execution by the <code>Inline</code> statement. The end of the text is assumed to be the first occurrence of the words <code>End Inline</code> appearing on a line.
----------------	--

Example

```
Sub Main()
    Inline MacScript
        -- This is an AppleScript comment.
        Beep
        Display Dialog "AppleScript" buttons "OK" default button "OK"
        Display Dialog Current Date
    End Inline
End Sub
```

See Also MacScript (statement).

Platform(s) Windows and Macintosh.

Input# (statement)

Syntax `Input [#]filename%, variable[, variable]...`

Description Reads data from the file referenced by *filename* into the given variables.

Comments Each *variable* must be type-matched to the data in the file. For example, a `String` variable must be matched to a string in the file.

The following parsing rules are observed while reading each variable in the variable list:

1. Leading white space is ignored (spaces and tabs).

2. When reading `String` variables, if the first character on the line is a quotation mark, then characters are read up to the next quotation mark or the end of the line, whichever comes first. Blank lines are read as empty strings. If the first character read is not a quotation mark, then characters are read up to the first comma or the end of the line, whichever comes first. String delimiters (quotes, comma, end-of-line) are not included in the returned string.
3. When reading numeric variables, scanning of the number stops when the first nonnumber character (such as a comma, a letter, or any other unexpected character) is encountered. Numeric errors are ignored while reading numbers from a file. The resultant number is automatically converted to the same type as the variable into which the value will be placed. If there is an error in conversion, then 0 is stored into the variable.

After reading the number, input is skipped up to the next delimiter—a comma, an end-of-line, or an end-of-file.

Numbers must adhere to any of the following syntaxes:

`[- | +]digits [. digits] [E [- | +] digits] [! | # | % | & | @]`

`&Hhexdigits [! | # | % | &]`

`&[O]octaldigits [! | # | % | & | @]`

4. When reading `Boolean` variables, the first character must be `#`; otherwise, a runtime error occurs. If the first character is `#`, then input is scanned up to the next delimiter (a comma, an end-of-line, or an end-of-file). If the input matches `#FALSE#`, then `False` is stored in the `Boolean`; otherwise `True` is stored.
5. When reading `Date` variables, the first character must be `#`; otherwise, a runtime error occurs. If the first character is `#`, then the input is scanned up to the next delimiter (a comma, an end-of-line, or an end-of-file). If the input ends in a `#` and the text between the `#`'s can be correctly interpreted as a date, then the date is stored; otherwise, December 31, 1899, is stored.

Normally, dates that follow the universal date format are input from sequential files. These dates use this syntax:

`#YYYY-MM-DD HH:MM:SS#`

where *YYYY* is a year between 100 and 9999, *MM* is a month between 1 and 12, *DD* is a day between 1 and 31, *HH* is an hour between 0 and 23, *MM* is a minute between 0 and 59, and *SS* is a second between 0 and 59.

6. When reading `Variant` variables, if the data begins with a quotation mark, then a string is read consisting of the characters between the opening quotation mark and the closing quotation mark, end-of-line, or end-of-file.

If the input does not begin with a quotation mark, then input is scanned up to the next comma, end-of-line, or end-of-file and a determination is made as to what data is being represented. If the data cannot be represented as a number, `Date`, `Error`, `Boolean`, or `Null`, then it is read as a string.

The following table describes how special data is interpreted as variants:

Blank line	Read as an <code>Empty</code> variant.
<code>#NULL#</code>	Read as a <code>Null</code> variant.
<code>#TRUE#</code>	Read as a <code>Boolean</code> variant.
<code>#FALSE#</code>	Read as a <code>Boolean</code> variant.
<code>#ERROR code#</code>	Read as a user-defined error.
<code>#date#</code>	Read as a <code>Date</code> variant.
<code>"text"</code>	Read as a <code>String</code> variant.

If an error occurs in interpretation of the data as a particular type, then that data is read as a `String` variant.

When reading numbers into variants, the optional type-declaration character determines the `VarType` of the resulting variant. If no type-declaration character is specified, then WM Basic will read the number according to the following rules:

Rule 1: If the number contains a decimal point or an exponent, then the number is read as `Currency`. If there is an error converting to `Currency`, then the number is treated as a `Double`.

Rule 2: If the number does not contain a decimal point or an exponent, then the number is stored in the smallest of the following data types that most accurately represents that value: `Integer`, `Long`, `Currency`, `Double`.

7. End-of-line is interpreted as either a single line feed, a single carriage return, or a carriage-return/line-feed pair. Thus, text files from any platform can be interpreted using this command.

The *filenumber* parameter is a number that is used by WM Basic to refer to the open file the number passed to the Open statement.

The *filenumber* must reference a file opened in Input mode. It is good practice to use the Write statement to write data elements to files read with the Input statement to ensure that the variable list is consistent between the input and output routines.

Example 'This example creates a file called test.dat and writes a series of 'variables into it. Then the variables are read using the Input# 'function.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Open "test.dat" For Output As #1
    Write #1,2112,"David","McCue","123-45-6789"
    Close

    Open "test.dat" For Input As #1
    Input #1,x%,st1$,st2$,st3$
    msg = "Employee " & x% & " Information" & crlf & crlf
    msg = msg & "First Name: " & st1$ & crlf
    msg = msg & "Last Name: " & st2$ & crlf
    msg = msg & "Social Security Number: " & st3$
    MsgBox msg
    Close

    Kill "test.dat"
End Sub
```

See Also Open (statement); Get (statement); Line Input# (statement); Input, Input\$ (functions).

Platform(s) Windows and Macintosh.

Input, Input\$ (functions)

Syntax Input[\$] (*numbytes* , [#] *filenumber*)

Description Returns *numbytes* characters read from a given sequential file.

Comments Input\$ returns a String, whereas Input returns a String variant.

The Input/Input\$ functions require the following parameters:

Parameter	Description
<i>numbytes</i>	Integer containing the number of bytes to be read from the file.
<i>filenumber</i>	Integer referencing a file opened in either Input or Binary mode. This is the same number passed to the Open statement.

This function reads all characters, including spaces and end-of-lines.

Example 'This example opens the autoexec.bat file and displays it in a 'dialog box.

```
Const crlf = Chr$(13) & Chr$(10)

Sub Main()
    x& = FileLen("c:\autoexec.bat")
    If x& > 0 Then
        Open "c:\autoexec.bat" For Input As #1
    Else
        MsgBox "File not found or empty."
        Exit Sub
    End If

    If x& > 80 Then
        ins = Input(80,#1)
    Else
        ins = Input(x,#1)
    End If
    Close
    MsgBox "File length: " & x& & crlf & ins
End Sub
```

See Also Open (statement); Get (statement); Input# (statement); Line Input# (statement).

Platform(s) Windows and Macintosh.

InputBox, InputBox\$ (functions)

Syntax InputBox[\$](prompt [, [title] [, [default] [, X,Y]])

Description Displays a dialog box with a text box into which the user can type.

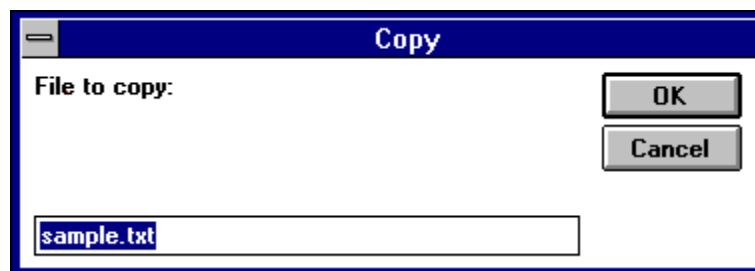
Comments The content of the text box is returned as a `String` (in the case of `InputBox$`) or as a `String` variant (in the case of `InputBox`). A zero-length string is returned if the user selects `Cancel`.

The `InputBox`/`InputBox$` functions take the following parameters:

Parameter	Description
<i>prompt</i>	Text to be displayed above the text box. The <i>prompt</i> parameter can contain multiple lines, each separated with an end-of-line (a carriage return, line feed, or carriage-return/line-feed pair). A runtime error is generated if <i>prompt</i> is <code>Null</code> .
<i>title</i>	Caption of the dialog box. If this parameter is omitted, then no title appears as the dialog box's caption. A runtime error is generated if <i>title</i> is <code>Null</code> .
<i>default</i>	Default response. This string is initially displayed in the text box. A runtime error is generated if <i>default</i> is <code>Null</code> .
<i>X, Y</i>	Integer coordinates, given in twips (twentieths of a point), specifying the upper left corner of the dialog box relative to the upper left corner of the screen. If the position is omitted, then the dialog box is positioned on or near the application executing the script.

Example

```
Sub Main()  
    s$ = InputBox$ ("File to copy:", "Copy", "sample.txt")  
End Sub
```



See Also `MsgBox` (statement); `AskBox$` (function); `AskPassword$` (function); `OpenFilename$` (function); `SaveFilename$` (function); `SelectBox` (function); `AnswerBox` (function).

Platform(s) Windows and Macintosh.

InStr (function)

Syntax `InStr([start,] search, find [,compare])`

Description Returns the first character position of string *find* within string *search*.

Comments The `InStr` function takes the following parameters:

Parameter	Description						
<i>start</i>	Integer specifying the character position where searching begins. The <i>start</i> parameter must be between 1 and 32767. If this parameter is omitted, then the search starts at the beginning (<i>start</i> = 1).						
<i>search</i>	Text to search. This can be any expression convertible to a <code>String</code> .						
<i>find</i>	Text for which to search. This can be any expression convertible to a <code>String</code> .						
<i>compare</i>	Integer controlling how string comparisons are performed: <table> <tr> <td>0</td><td>String comparisons are case-sensitive.</td></tr> <tr> <td>1</td><td>String comparisons are case-insensitive.</td></tr> <tr> <td>Any other value</td><td>A runtime error is produced.</td></tr> </table> If this parameter is omitted, then string comparisons use the current <code>Option Compare</code> setting. If no <code>Option Compare</code> statement has been encountered, then <code>Binary</code> is used (i.e., string comparisons are case-sensitive). If the string is found, then its character position within <i>search</i> is returned, with 1 being the character position of the first character. If <i>find</i> is not found, or <i>start</i> is greater than the length of <i>search</i> , or <i>search</i> is zero-length, then 0 is returned.	0	String comparisons are case-sensitive.	1	String comparisons are case-insensitive.	Any other value	A runtime error is produced.
0	String comparisons are case-sensitive.						
1	String comparisons are case-insensitive.						
Any other value	A runtime error is produced.						

Example 'This example checks to see whether one string is in another and, 'if it is, then it copies the string to a variable and displays the 'result.

```
Sub Main()
    a$ = "This string contains the name Stuart and other characters."
    x% = InStr(a$, "Stuart", 1)
    If x% <> 0 Then
        b$ = Mid$(a$, x%, 6)
        MsgBox b$ & " was found."
        Exit Sub
    Else
        MsgBox "Stuart not found."
    End If
End Sub
```

See Also `Mid`, `Mid$` (functions); `Option Compare` (statement); `Item$` (function); `Word$` (function); `Line$` (function).

Platform(s) Windows and Macintosh.

Int (function)

Syntax `Int (number)`

Description Returns the integer part of *number*.

Comments This function returns the integer part of a given value by returning the first integer less than the *number*. The sign is preserved.

The `Int` function returns the same type as *number*, with the following exceptions:

- If *number* is `Empty`, then an `Integer` variant of value 0 is returned.
- If *number* is a `String`, then a `Double` variant is returned.
- If *number* is `Null`, then a `Null` variant is returned.

Example 'This example extracts the integer part of a number.

```
Sub Main()  
  a# = -1234.5224  
  b% = Int(a#)  
  MsgBox "The integer part of -1234.5224 is: " & b%  
End Sub
```

See Also `Fix` (function); `CInt` (function).

Platform(s) Windows and Macintosh.

Integer (data type)

Syntax `Integer`

Description A data type used to declare whole numbers with up to four digits of precision.

Comments `Integer` variables are used to hold numbers within the following range:

`-32768 <= integer <= 32767`

Internally, integers are 2-byte short values. Thus, when appearing within a structure, integers require 2 bytes of storage. When used with binary or random files, 2 bytes of storage are required.

When passed to external routines, `Integer` values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack.

The type-declaration character for `Integer` is `%`.

See Also `Currency` (data type); `Date` (data type); `Double` (data type); `Long` (data type); `Object` (data type); `Single` (data type); `String` (data type); `Variant` (data type); `Boolean` (data type); `DefType` (statement); `CInt` (function).

Platform(s) Windows and Macintosh.

IPmt (function)

Syntax IPmt(*Rate*, *Per*, *Nper*, *Pv*, *Fv*, *Due*)

Description Returns the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Comments An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages, monthly savings plans, and retirement plans.

The following table describes the different parameters:

Parameter	Description
<i>Rate</i>	Double representing the interest rate per period. If the payment periods are monthly, be sure to divide the annual interest rate by 12 to get the monthly rate.
<i>Per</i>	Double representing the payment period for which you are calculating the interest payment. If you want to know the interest paid or received during period 20 of an annuity, this value would be 20.
<i>Nper</i>	Double representing the total number of payments in the annuity. This is usually expressed in months, and you should be sure that the interest rate given above is for the same period that you enter here.
<i>Pv</i>	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan because that is the amount of cash you have in the present. In the case of a retirement plan, this value would be the current value of the fund because you have a set amount of principal in the plan.
<i>Fv</i>	Double representing the future value of your annuity. In the case of a loan, the future value would be zero because you will have paid it off. In the case of a savings plan, the future value would be the balance of the account after all payments are made.
<i>Due</i>	Integer indicating when payments are due. If this parameter is 0, then payments are due at the end of each period (usually, the end of the month). If this value is 1, then payments are due at the start of each period (the beginning of the month).

Rate and *Nper* must be expressed in the same units. If *Rate* is expressed in percentage paid per month, then *Nper* must also be expressed in months. If *Rate* is an annual rate, then the period given in *Nper* should also be in years or the annual *Rate* should be divided by 12 to obtain a monthly rate.

If the function returns a negative value, it represents interest you are paying out, whereas a positive value represents interest paid to you.

Example 'This example calculates the amount of interest paid on a \$1,000.00
'loan financed over 36 months with an annual interest rate of 10%.
'Payments are due at the beginning of the month. The interest paid
'during the first 10 months is displayed in a table.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    For x = 1 to 10
        ipm# = IPmt((.10/12),x,36,1000,0,1)
        msg = msg & Format(x,"00") & " : " & Format(ipm#," 0,0.00") &
crlf
    Next x
    MsgBox msg
End Sub
```

See Also NPer (function); Pmt (function); PPmt (function); Rate (function).

Platform(s) Windows and Macintosh.

IRR (function)

Syntax IRR (*ValueArray()*, *Guess*)

Description Returns the internal rate of return for a series of periodic payments and receipts.

Comments The internal rate of return is the equivalent rate of interest for an investment consisting of a series of positive and/or negative cash flows over a period of regular intervals. It is usually used to project the rate of return on a business investment that requires a capital investment up front and a series of investments and returns on investment over time.

The IRR function requires the following parameters:

Parameter	Description
<i>ValueArray()</i>	<p>Array of <code>Double</code> numbers that represent payments and receipts. Positive values are payments, and negative values are receipts.</p> <p>There must be at least one positive and one negative value to indicate the initial investment (negative value) and the amount earned by the investment (positive value).</p>
<i>Guess</i>	<p><code>Double</code> containing your guess as to the value that the IRR function will return. The most common guess is .1 (10 percent).</p> <p>The value of IRR is found by iteration. It starts with the value of <i>Guess</i> and cycles through the calculation adjusting <i>Guess</i> until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, IRR fails, and the user must pick a better guess.</p>

Example 'This example illustrates the purchase of a lemonade stand for \$800
'and a series of incomes from the sale of lemonade over 12 months.
'The projected incomes for this example are generated in two
'For...Next Loops, and then the internal rate of return is calculated
'and displayed. (Not a bad investment!)

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim valu#(12)
    valu(1) = -800                                'Initial investment
    msg = valu(1) & ", "

    'Calculate the second through fifth months' sales.
    For x = 2 To 5
        valu(x) = 100 + (x * 2)
        msg = msg & valu(x) & ", "
    Next x

    'Calculate the sixth through twelfth months' sales.
    For x = 6 To 12
        valu(x) = 100 + (x * 10)
        msg = msg & valu(x) & ", "
    Next x

    'Calculate the equivalent investment return rate.
    retn# = IRR(valu,.1)
    msg = "The values: " & crlf & msg & crlf & crlf
    MsgBox msg & "Return rate: " & Format(retn#,"Percent")
End Sub
```

See Also Fv (function); MIRR (function); Npv (function); Pv (function).

Platform(s) Windows and Macintosh.

Is (operator)

Syntax *object* Is [*object* | Nothing]

Description Returns True if the two operands refer to the same object; returns False otherwise.

Comments This operator is used to determine whether two object variables refer to the same object. Both operands must be object variables of the same type (i.e., the same data object type or both of type Object).

The Nothing constant can be used to determine whether an object variable is uninitialized:

```
If MyObject Is Nothing Then MsgBox "MyObject is uninitialized."
```

Uninitialized object variables reference no object.

Example 'This function inserts the date into a Microsoft Word document.

```
Sub InsertDate(ByVal WinWord As Object)
    If WinWord Is Nothing Then
        MsgBox "Object variant is not set."
    Else
        WinWord.Insert Date$
    End If
End Sub

Sub Main()
    Dim WinWord As Object
    On Error Resume Next
    WinWord = CreateObject("word.basic")
    InsertDate WinWord
End Sub
```

See Also Operator Precedence (topic); Like (operator).

Platform(s) Windows and Macintosh.

Platform When comparing OLE automation objects, the **Is** operator will only return

Notes: True if the operands reference the same OLE automation object. This is different from data objects. For example, the following use of **Is** (using the object class called `excel.application`) returns True:

Windows,
Macintosh

```
Dim a As Object
Dim b As Object
a = CreateObject("excel.application")
b = a
If a Is b Then Beep
```

The following use of **Is** will return False, even though the actual objects may be the same:

```
Dim a As Object
Dim b As Object
a = CreateObject("excel.application")
b = GetObject(,"excel.application")
If a Is b Then Beep
```

The **Is** operator may return False in the above case because, even though a and b reference the same object, they may be treated as different objects by OLE 2.0 (this is dependent on the OLE 2.0 server application).

IsDate (function)

Syntax `IsDate(expression)`

Description Returns True if *expression* can be legally converted to a date; returns False otherwise.

Example

```

Sub Main()
    Dim a As Variant
Retry:
    a = InputBox("Enter a date.", "Enter Date")
    If IsDate(a) Then
        MsgBox Format(a,"long date")
    Else
        MsgBox "Not quite, please try again!"
        Goto Retry
    End If
End Sub

```

See Also Variant (data type); IsEmpty (function); IsError (function); IsObject (function); VarType (function); IsNull (function).

Platform(s) Windows and Macintosh.

IsEmpty (function)

Syntax IsEmpty(*expression*)

Description Returns True if *expression* is a Variant variable that has never been initialized; returns False otherwise.

Comments The IsEmpty function is the same as the following:

```
(VarType(expression) = vbEmpty)
```

Example

```

Sub Main()
    Dim a As Variant
    If IsEmpty(a) Then
        a = 1.0#           'Give uninitialized data a Double value 0.0.
        MsgBox "The variable has been initialized to: " & a
    Else
        MsgBox "The variable was already initialized!"
    End If
End Sub

```

See Also Variant (data type); IsDate (function); IsError (function); IsObject (function); VarType (function); IsNull (function).

Platform(s) Windows and Macintosh.

IsError (function)

Syntax IsError(*expression*)

Description Returns True if *expression* is a user-defined error value; returns False otherwise.

Example 'This example creates a function that divides two numbers. If there is an error dividing the numbers, then a variant of type "error" is returned. Otherwise, the function returns the result of the division. The IsError function is used to determine whether the function encountered an error.

```
Function Div(ByVal a,ByVal b) As Variant
    If b = 0 Then
        Div = CVErr(2112)          'Return a special error value.
    Else
        Div = a / b                'Return the division.
    End If
End Function

Sub Main()
    Dim a As Variant
    a = Div(10,12)
    If IsError(a) Then
        MsgBox "The following error occurred: " & CStr(a)
    Else
        MsgBox "The result is: " & a
    End If
End Sub
```

See Also Variant (data type); IsEmpty (function); IsDate (function); IsObject (function); VarType (function); IsNull (function).

Platform(s) Windows and Macintosh.

IsMissing (function)

Syntax IsMissing(*variable*)

Description Returns True if *variable* was passed to the current subroutine or function; returns False if omitted.

Comments The IsMissing is used with variant variables passed as optional parameters (using the Optional keyword) to the current subroutine or function. For non-variant variables or variables that were not declared with the Optional keyword, IsMissing will always return True.

Example 'The following function runs an application and optionally minimizes it. If 'the optional isMinimize parameter is not specified by the caller, then the 'application is not minimized.

```
Sub Test(AppName As String,Optional isMinimize As Variant)
    app = Shell(AppName)
    If Not IsMissing(isMinimize) Then
        AppMinimize app
    Else
        AppMaximize app
    End If
End Sub

Sub Main
    Test "Notepad"           'Maximize this application
    Test "Notepad",True      'Mimimize this application
End Sub
```

See Also Declare (statement), Sub...End Sub (statement), Function...End Function (statement)

Platform(s) Windows and Macintosh.

IsNull (function)

Syntax IsNull(*expression*)

Description Returns True if *expression* is a Variant variable that contains no valid data; returns False otherwise.

Comments The IsNull function is the same as the following:

```
(VarType(expression) = vbNull)
```

Example

```
Sub Main()
    Dim a As Variant      'Initialized as Empty
    If IsNull(a) Then MsgBox "The variable contains no valid data."
    a = Empty * Null
    If IsNull(a) Then MsgBox "Null propagated through the expression."
End Sub
```

See Also Empty (constant); Variant (data type); IsEmpty (function); IsDate (function); IsError (function); IsObject (function); VarType (function).

Platform(s) Windows and Macintosh.

IsNumeric (function)

Syntax IsNumeric(*expression*)

Description Returns True if *expression* can be converted to a number; returns False otherwise.

Comments If passed a number or a variant containing a number, then `IsNumeric` always returns `True`.

If a `String` or `String` variant is passed, then `IsNumeric` will return `True` only if the string can be converted to a number. The following syntaxes are recognized as valid numbers:

```
&Hhexdigits[&|%|!|#|@]
&[O]octaldigits[&|%|!|#|@]
[-|+]digits[.digits][E[-|+]digits][!|%|&|#|@]
```

If an `Object` variant is passed, then the default property of that object is retrieved and one of the above rules is applied.

`IsNumeric` returns `False` if *expression* is a `Date`.

Example

```
Sub Main()
    Dim s$ As String
    s$ = InputBox("Enter a number.", "Enter Number")

    If IsNumeric(s$) Then
        MsgBox "You did good!"
    Else
        MsgBox "You didn't do so good!"
    End If
End Sub
```

See Also `Variant` (data type); `IsEmpty` (function); `IsDate` (function); `IsError` (function); `IsObject` (function); `VarType` (function); `IsNull` (function).

Platform(s) Windows and Macintosh.

IsObject (function)

Syntax `IsObject(expression)`

Description Returns `True` if *expression* is a `Variant` variable containing an `Object`; returns `False` otherwise.

Example 'This example will attempt to find a running copy of Excel and create 'a Excel object that can be referenced as any other object in 'WM Basic.

```
Sub Main()
    Dim v As Variant
    On Error Resume Next
    Set v = GetObject(,"Excel.Application")

    If IsObject(v) Then
        MsgBox "The default object value is: " & v = v.Value      'Access
value property of the object.
    Else
        MsgBox "Excel not loaded."
    End If
End Sub
```

See Also Variant (data type); IsEmpty (function); IsDate (function); IsError (function); VarType (function); IsNull (function).

Platform(s) Windows and Macintosh.

Item\$ (function)

Syntax Item\$(text\$,first,last [,delimiters\$])

Description Returns all the items between *first* and *last* within the specified formatted text list.

Comments The Item\$ function takes the following parameters:

Parameter	Description
<i>text\$</i>	String containing the text from which a range of items is returned.
<i>first</i>	Integer containing the index of the first item to be returned. If <i>first</i> is greater than the number of items in <i>text\$</i> , then a zero-length string is returned.
<i>last</i>	Integer containing the index of the last item to be returned. All of the items between <i>first</i> and <i>last</i> are returned. If <i>last</i> is greater than the number of items in <i>text\$</i> , then all items from <i>first</i> to the end of text are returned.
<i>delimiters\$</i>	String containing different item delimiters. By default, items are separated by commas and end-of-lines. This can be changed by specifying different delimiters in the <i>delimiters\$</i> parameter.

Example 'This example creates two delimited lists and extracts a range from 'each, then displays the result in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    ilist$ = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15"
    slist$ = "1/2/3/4/5/6/7/8/9/10/11/12/13/14/15"
    list1$ = Item$(ilist$,5,12)
    list2$ = Item$(slist$,2,9,"/")
    MsgBox "The returned lists are: " & crlf & list1$ & crlf & list2$
End Sub
```

See Also ItemCount (function); Line\$ (function); LineCount (function); Word\$ (function); WordCount (function).

Platform(s) Windows and Macintosh.

ItemCount (function)

Syntax ItemCount(text\$ [,delimiters\$])

Description Returns an Integer containing the number of items in the specified delimited text.

Comments Items are substrings of a delimited text string. Items, by default, are separated by commas and/or end-of-lines. This can be changed by specifying different delimiters in the *delimiters\$* parameter. For example, to parse items using a backslash:

```
n = ItemCount(text$,"\")
```

Example 'This example creates two delimited lists and then counts the number 'of items in each. The counts are displayed in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    ilist$ = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15"
    slist$ = "1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/16/17/18/19"

    l1% = ItemCount(ilist$)
    l2% = ItemCount(slist$,"/")
    msg = "The first lists contains: " & l1% & " items." & crlf
    msg = msg & "The second list contains: " & l2% & " items."
    MsgBox msg
End Sub
```

See Also Item\$ (function); Line\$ (function); LineCount (function); Word\$ (function); WordCount (function).

Platform(s) Windows and Macintosh.

Keywords (topic)

A keyword is any word or symbol recognized by WM Basic as part of the language. All of the following are keywords:

- Built-in subroutine names, such as `MsgBox` and `Print`.
- Built-in function names, such as `Str$`, `Cdbl`, and `Mid$`.
- Special keywords, such as `To`, `Next`, `Case`, and `Binary`.
- Names of any extended language elements.

Restrictions

All keywords are reserved by WM Basic, in that you cannot create a variable, function, constant, or subroutine with the same name as a keyword. However, you are free to use all keywords as the names of structure members.

Platform(s) Windows and Macintosh.

Kill (statement)

Syntax `Kill filespec$`

Description Deletes all files matching *filespec\$*.

Comments The *filespec\$* argument can include wildcards, such as * and ?. The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complex searching patterns. The following table shows some examples.

This Pattern	Matches These Files	Doesn't Match These Files
S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT
C*T.TXT	CAT.TXT	CAP.TXT ACATS.TXT
C*T	CAT	CAT.DOC CAP.TXT
C?T	CAT CUT	CAT.TXT CAPIT CT
*	(All files)	

Example 'This example looks to see whether file test1.dat exists. If it does not, then it creates both test1.dat and test2.dat. The existence of the files is tested again; if they exist, a message is generated, and then they are deleted. The final test looks to see whether they are still there and displays the result.

```
Sub Main()
  If Not FileExists("test1.dat") Then
    Open "test1.dat" For Output As #1
    Open "test2.dat" For Output As #2
    Close
  End If

  If FileExists ("test1.dat") Then
    MsgBox "File test1.dat exists."
    Kill "test?.dat"
  End If

  If FileExists ("test1.dat") Then
    MsgBox "File test1.dat still exists."
  Else
    MsgBox "test?.dat sucessfully deleted."
  End If
End Sub
```

See Also Name (statement).

Platform(s) Windows and Macintosh.

Platform Notes: Notice that WM Basic's filename matching is different than DOS's. The pattern "`*.*`" under DOS matches all files. With WM Basic, this pattern matches only files that have file extensions.

This function behaves the same as the "del" command in DOS.

Platform Notes: The Macintosh does not support wildcard characters such as `*` and `?`. These are valid filename characters. Instead of wildcards, the Macintosh uses the `MacID` function to specify a collection of files of the same type. The syntax for this function is:

```
Kill MacID(text$)
```

The *text*\$ parameter is a four-character string containing a file type, a resource type, an application signature, or an Apple event. A runtime error occurs if the `MacID` function is used on platforms other than the Macintosh.

LBound (function)

Syntax `LBound(ArrayVariable () [, dimension])`

Description Returns an Integer containing the lower bound of the specified dimension of the specified array variable.

Comments The *dimension* parameter is an integer specifying the desired dimension. If this parameter is not specified, then the lower bound of the first dimension is returned.

The `LBound` function can be used to find the lower bound of a dimension of an array returned by an OLE automation method or property:

```
LBound(object.property [ , dimension ] )
```

```
LBound(object.method [ , dimension ] )
```

Examples

```
Sub Main()
    'This example dimensions two arrays and displays their lower
    bounds.

    Dim a(5 To 12)
    Dim b(2 To 100, 9 To 20)

    lba = LBound(a)
    lbb = LBound(b,2)
    MsgBox "The lower bound of a is: " & lba & " The lower bound of b
    is: " & lbb

    'This example uses LBound and UBound to dimension a dynamic array
    to
    'hold a copy of an array redimmed by the FileList statement.

    Dim fl$()
    FileList fl$,"*.*"
    count = UBound(fl$)
    If ArrayDims(a) Then
        Redim nl$(LBound(fl$) To UBound(fl$))
        For x = 1 To count
            nl$(x) = fl$(x)
        Next x
        MsgBox "The last element of the new array is: " & nl$(count)
    End If
End Sub
```

See Also UBound (function); ArrayDims (function); Arrays (topic).

Platform(s) Windows and Macintosh.

LCase, LCase\$ (functions)

Syntax LCase[\$](*text*)

Description Returns the lowercase equivalent of the specified string.

Comments LCase\$ returns a String, whereas LCase returns a String variant.

Null is returned if *text* is Null.

Example 'This example shows the LCase function used to change uppercase names
'to lowercase with an uppercase first letter.

```
Sub Main()
    lname$ = "WILLIAMS"
    fl$ = Left$(lname$,1)
    rest$ = Mid$(lname$,2,Len(lname$))
    lname$ = fl$ & LCase$(rest$)
    MsgBox "The converted name is: " & lname$
End Sub
```

See Also UCase, UCase\$ (functions).

Platform(s) Windows and Macintosh.

Left, Left\$ (functions)

Syntax `Left[$](text,NumChars)`

Description Returns the leftmost *NumChars* characters from a given string.

Comments `Left$` returns a `String`, whereas `Left` returns a `String` variant.

NumChars is an `Integer` value specifying the number of character to return. If *NumChars* is 0, then a zero-length string is returned. If *NumChars* is greater than or equal to the number of characters in the specified string, then the entire string is returned.

`Null` is returned if *text* is `Null`.

Example 'This example shows the `Left$` function used to change uppercase names to lowercase with an uppercase first letter.

```
Sub Main()
    lname$ = "WILLIAMS"
    fl$ = Left$(lname$,1)
    rest$ = Mid$(lname$,2,Len(lname$))
    lname$ = fl$ & LCase$(rest$)
    MsgBox "The converted name is: " & lname$
End Sub
```

See Also `Right`, `Right$` (functions).

Platform(s) Windows and Macintosh.

Len (function)

Syntax `Len(expression)`

Description Returns the number of characters in *expression* or the number of bytes required to store the specified variable.

Comments If *expression* evaluates to a string, then `Len` returns the number of characters in a given string or 0 if the string is empty. When used with a `Variant` variable, the length of the variant when converted to a `String` is returned. If *expression* is a `Null`, then `Len` returns a `Null` variant.

If used with a non-`String` or non-`Variant` variable, the function returns the number of bytes occupied by that data element.

When used with user-defined data types, the function returns the combined size of each member within the structure. Since variable-length strings are stored elsewhere, the size of each variable-length string within a structure is 2 bytes.

The following table describes the sizes of the individual data elements:

Data Element	Size
Integer	2 bytes.
Long	4 bytes.
Float	4 bytes.
Double	8 bytes.
Currency	8 bytes.
String (variable-length)	Number of characters in the string.
String (fixed-length)	The length of the string as it appears in the string's declaration.
Objects	0 bytes. Both data object variables and variables of type <code>Object</code> are always returned as 0 size.
User-defined type	Combined size of each structure member. Variable-length strings within structures require 2 bytes of storage. Arrays within structures are fixed in their dimensions. The elements for fixed arrays are stored within the structure and therefore require the number of bytes for each array element multiplied by the size of each array dimension:

$$\text{element_size} * \text{dimension1} * \text{dimension2} \dots$$

The `Len` function always returns 0 with object variables or any data object variable.

Examples

```

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    'This example shows the Len function used in a routine to change
    'uppercase names to lowercase with an uppercase first letter.

    lname$ = "WILLIAMS"
    fl$ = Left$(lname$,1)
    ln% = Len(lname$)
    rest$ = Mid$(lname$,2,ln%)
    lname$ = fl$ & LCase$(rest$)
    MsgBox "The converted name is: " & lname$

    'This example returns a table of lengths for standard numeric
    types.

    Dim lns(4)
    a% = 100 : b& = 200 : c! = 200.22 : d# = 300.22
    lns(1) = Len(a%)
    lns(2) = Len(b&)
    lns(3) = Len(c!)
    lns(4) = Len(d#)
    msg = "Lengths of standard types:" & crlf
    msg = msg & "Integer: " & lns(1) & crlf
    msg = msg & "Long: " & lns(2) & crlf
    msg = msg & "Single: " & lns(3) & crlf
    msg = msg & "Double: " & lns(4) & crlf
    MsgBox msg
End Sub

```

See Also InStr (function).

Platform(s) Windows and Macintosh.

Let (statement)

Syntax [Let] *variable* = *expression*

Description Assigns the result of an expression to a variable.

Comments The use of the word `Let` is supported for compatibility with other implementations of WM Basic. Normally, this word is dropped.

When assigning expressions to variables, internal type conversions are performed automatically between any two numeric quantities. Thus, you can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This happens when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error:

```
Dim amount As Long
Dim quantity As Integer

amount = 400123      'Assign a value out of range for int.
quantity = amount    'Attempt to assign to Integer.
```

When performing an automatic data conversion, underflow is not an error.

Example

```
Sub Main()
    Let a$ = "This is a string."
    Let b% = 100
    Let c# = 1213.3443
End Sub
```

See Also = (keyword); Expression Evaluation (topic).

Platform(s) Windows and Macintosh.

Like (operator)

Syntax *expression Like pattern*

Description Compares two strings and returns `True` if the *expression* matches the given *pattern*; returns `False` otherwise.

Comments Case sensitivity is controlled by the `Option Compare` setting.

The *pattern* expression can contain special characters that allow more flexible matching:

Character	Evaluates To
<code>?</code>	Matches a single character.
<code>*</code>	Matches one or more characters.
<code>#</code>	Matches any digit.
<code>[range]</code>	Matches if the character in question is within the specified range.
<code>[!range]</code>	Matches if the character in question is not within the specified range.

A *range* specifies a grouping of characters. To specify a match of any of a group of characters, use the syntax [ABCDE]. To specify a range of characters, use the syntax [A-Z]. Special characters must appear within brackets, such as []*?#.

If *expression* or *pattern* is not a string, then both *expression* and *pattern* are converted to String variants and compared, returning a Boolean variant. If either variant is Null, then Null is returned.

The following table shows some examples:

<i>expression</i>	True If pattern Is	False If pattern Is
"EBW"	"E*W", "E*"	"E*B"
"Version"	"V[e]?s*n"	"V[r]?s*N"
"2.0"	"#.#", "#?#" "	"###", "#?[!0-9]" "
"[ABC]"	"[[]*"	"[ABC]", "[*]" "

Example 'This example demonstrates various uses of the Like function.

```
Sub Main()
    a$ = "This is a string variable of 123456 characters"
    b$ = "123.45"
    If a$ Like "[A-Z][g-i]*" Then MsgBox "The first comparison is
True."
    If b$ Like "##3.##" Then MsgBox "The second comparison is True."
    If a$ Like "**variable*" Then MsgBox "The third comparison is True."
End Sub
```

See Also Operator Precedence (topic); Is (operator); Option Compare (statement).

Platform(s) Windows and Macintosh.

Line Input# (statement)

Syntax Line Input [#]filename, variable

Description Reads an entire line into the given variable.

Comments The *filenumber* parameter is a number that is used by WM Basic to refer to the open file the number passed to the Open statement. The *filenumber* must reference a file opened in Input mode.

The file is read up to the next end-of-line, but the end-of-line character(s) is (are) not returned in the string. The file pointer is positioned after the terminating end-of-line.

The *variable* parameter is any string or variant variable reference. This statement will automatically declare the variable if the specified variable has not yet been used or dimensioned.

This statement recognizes either a single line feed or a carriage-return/line-feed pair as the end-of-line delimiter.

Example 'This example reads five lines of the autoexec.bat file and displays them in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Open "c:\autoexec.bat" For Input As #1
    For x = 1 To 5
        Line Input #1,lin$
        msg = msg & lin$ & crlf
    Next x
    MsgBox "The first 5 lines of your autoexec.bat are:" & crlf & msg
End Sub
```

See Also Open (statement); Get (statement); Input# (statement); Input, Input\$ (functions).

Platform(s) Windows and Macintosh.

Line Numbers (topic)

Line numbers are not supported by WM Basic.

As an alternative to line numbers, you can use meaningful labels as targets for absolute jumps, as shown below:

```
Sub Main()
    Dim i As Integer
    On Error Goto MyErrorTrap

    i = 0
LoopTop:
    i = i + 1
    If i < 10 Then Goto LoopTop

MyErrorTrap:
    MsgBox "An error occurred."

End Sub
```

Line\$ (function)

Syntax `Line$(text$,first[,last])`

Description Returns a String containing a single line or a group of lines between *first* and *last*.

Comments Lines are delimited by carriage return, line feed, or carriage-return/line-feed pairs.

The `Line$` function takes the following parameters:

Parameter	Description
<i>text\$</i>	String containing the text from which the lines will be extracted.
<i>first</i>	Integer representing the index of the first line to return. If <i>last</i> is omitted, then this line will be returned. If <i>first</i> is greater than the number of lines in <i>text\$</i> , then a zero-length string is returned.
<i>last</i>	Integer representing the index of the last line to return.

Example 'This example reads five lines of the `autoexec.bat` file, extracts the 'third and fourth lines with the `Line$` function, and displays them in a 'dialog box.

```
Const crlf = Chr$(13) + Chr$(10)
Sub Main()
    Open "c:\autoexec.bat" For Input As #1
    For x = 1 To 5
        Line Input #1,lin$
        txt = txt & lin$ & crlf
    Next x
    lines$ = Line$(txt,3,4)
    MsgBox lines$
End Sub
```

See Also `Item$ (function)`; `ItemCount (function)`; `LineCount (function)`; `Word$ (function)`; `WordCount (function)`.

Platform(s) Windows and Macintosh.

LineCount (function)

Syntax `LineCount(text$)`

Description Returns an Integer representing the number of lines in *text\$*.

Comments Lines are delimited by carriage return, line feed, or both.

Example 'This example reads the first ten lines of your autoexec.bat file,
'uses the LineCount function to determine the number of lines,
'and then displays them in a message box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    x = 1
    Open "c:\autoexec.bat" For Input As #1
    While (x < 10) And Not EOF(1)
        Line Input #1,lin$
        txt = txt & lin$ & crlf
        x = x + 1
    Wend
    lines! = LineCount(txt)
    MsgBox "The number of lines in txt is: " & lines! & crlf & crlf &
txt
End Sub
```

See Also Item\$ (function); ItemCount (function); Line\$ (function); Word\$ (function); WordCount (function).

Platform(s) Windows and Macintosh.

ListBox (statement)

Syntax `ListBox X,Y,width,height,ArrayVariable, .Identifier`

Description Creates a list box within a dialog box template.

Comments When the dialog box is invoked, the list box will be filled with the elements contained in *ArrayVariable*.

This statement can only appear within a dialog box template (i.e., between the `Begin Dialog` and `End Dialog` statements).

The `ListBox` statement requires the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>ArrayVariable</i>	Specifies a single-dimensioned array of strings used to initialize the elements of the list box. If this array has no dimensions, then the list box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. <i>ArrayVariable</i> can specify an array of any fundamental data type (structures are not allowed). <code>Null</code> and <code>Empty</code> values are treated as zero-length strings.
<i>.Identifier</i>	Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>). This parameter also creates an integer variable whose value corresponds to the index of the list box's selection (0 is the first item, 1 is the second, and so on). This variable can be accessed using the following syntax:

DialogVariable.Identifier

Example 'This example creates a dialog box with two list boxes, one containing files and the other containing directories.

```
Sub Main()
    Dim files() As String
    Dim dirs() As String
    Begin Dialog ListBoxTemplate 16,32,184,96,"Sample"
        Text 8,4,24,8,"&Files:"
        ListBox 8,16,60,72,files$, .Files
        Text 76,4,21,8,"&Dirs:"
        ListBox 76,16,56,72,dirs$, .Dirs
        OKButton 140,4,40,14
        CancelButton 140,24,40,14
    End Dialog
    FileList files
    FileDirs dirs

    Dim ListBoxDialog As ListBoxTemplate
    rc% = Dialog(ListBoxDialog)
End Sub
```

See Also CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Platform(s) Windows and Macintosh.

ListBoxEnabled (function)

Syntax `ListBoxEnabled(name$ | id)`

Description Returns `True` if the given list box is enabled within the active window or dialog box; returns `False` otherwise.

Comments This function is used to determine whether a list box is enabled within the current window or dialog box. If there is no active window, `False` will be returned.

The `ListBoxEnabled` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the list box. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the list box. Note: The <code>ListBoxEnabled</code> function is used to determine whether a list box is enabled in another application's dialog box. Use the <code>DlgEnable</code> function in dynamic dialog boxes.

Example 'This example checks to see whether the list box is enabled before
'setting the focus to it.

```
Sub Main()  
    If ListBoxEnabled("Files:") Then ActivateControl "Files:"  
End Sub
```

See Also GetListBoxItem\$ (function); GetListBoxItemCount (function);
ListBoxExists (function); SelectListBoxItem (statement).

Platform(s) Windows.

ListBoxExists (function)

Syntax `ListBoxExists(name$ | id)`

Description Returns `True` if the given list box exists within the active window or dialog box; returns `False` otherwise.

Comments This function is used to determine whether a list box exists within the current window or dialog box. If there is no active window, `False` will be returned.

The `ListBoxExists` function takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the list box. The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.
<i>id</i>	Integer specifying the ID of the list box.
Note: The <code>ListBoxExists</code> function is used to determine whether a list box exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example 'This example checks to see whether the list box exists and is enabled
'before setting the focus to it.

```
Sub Main()
    If ListBoxExists("Files:") Then
        If ListBoxEnabled("Files:") Then
            ActivateControl "Files:"
        End If
    End If
End Sub
```

See Also `GetListBoxItem$ (function)`; `GetListBoxItemCount (function)`;
`ListBoxEnabled (function)`; `SelectListBoxItem (statement)`.

Platform(s) Windows.

Literals (topic)

Literals are values of a specific type. The following table shows the different types of literals supported by WM Basic:

Literal	Description		
10	Integer whose value is 10.		
43265	Long whose value is 43,265.		
5#	Double whose value is 5.0. A number's type can be explicitly set using any of the following type-declaration characters:		
	<table> <tr> <td>%</td><td>Integer</td></tr> </table>	%	Integer
%	Integer		
	<table> <tr> <td>&</td><td>Long</td></tr> </table>	&	Long
&	Long		
	<table> <tr> <td>#</td><td>Double</td></tr> </table>	#	Double
#	Double		
	<table> <tr> <td>!</td><td>Single</td></tr> </table>	!	Single
!	Single		
5.5	Double whose value is 5.5. Any number with decimal point is considered a double.		
5.4E100	Double expressed in scientific notation.		
&HFF	Integer expressed in hexadecimal.		
&O47	Integer expressed in octal.		
&HFF#	Double expressed in hexadecimal.		
"hello"	String of five characters: hello.		
"" "hello" ""	String of seven characters: "hello". Quotation marks can be embedded within strings by using two consecutive quotation marks.		
#1/1/1994#	Date value whose internal representation is 34335.0. Any valid date can appear with #'s. Date literals are interpreted at execution time using the locale settings of the host environment. To ensure that date literals are correctly interpreted for all locales, use the international date format: <i>#YYYY-MM-DD HH:MM:SS#</i>		

Constant Folding

WM Basic supports constant folding where constant expressions are calculated by the compiler at compile time. For example, the expression

```
i% = 10 + 12
```

is the same as:

```
i% = 22
```

Similarly, with strings, the expression

```
s$ = "Hello," + " there" + Chr(46)
```

is the same as:

```
s$ = "Hello, there."
```

Loc (function)

Syntax `Loc (filenumber)`

Description Returns a Long representing the position of the file pointer in the given file.

Comments The *filenumber* parameter is an Integer used by WM Basic to refer to the number passed by the Open statement to WM Basic.

The Loc function returns different values depending on the mode in which the file was opened:

File Mode	Returns
Input	Current byte position divided by 128
Output	Current byte position divided by 128
Append	Current byte position divided by 128
Binary	Position of the last byte read or written
Random	Number of the last record read or written

Example 'This example reads 5 lines of the autoexec.bat file, determines the 'current location of the file pointer, and displays it in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Open "c:\autoexec.bat" For Input As #1
    For x = 1 To 5
        If Not EOF(1) Then Line Input #1,lin$
    Next x
    lc% = Loc(1)
    Close
    MsgBox "The file location is: " & lc%
End Sub
```

See Also Seek (function); Seek (statement); FileLen (function).

Platform(s) Windows and Macintosh.

Lock (statement)

Syntax Lock [#] *filename* [, { *record* | [*start*] To *end* }]

Description Locks a section of the specified file, preventing other processes from accessing that section of the file until the Unlock statement is issued.

Comments The Lock statement requires the following parameters:

Parameter	Description
<i>filename</i>	Integer used by WM Basic to refer to the open file—the number passed to the Open statement.
<i>record</i>	Long specifying which record to lock.
<i>start</i>	Long specifying the first record within a range to be locked.
<i>end</i>	Long specifying the last record within a range to be locked. For sequential files, the <i>record</i> , <i>start</i> , and <i>end</i> parameters are ignored. The entire file is locked. The section of the file is specified using one of the following:
Syntax	Description
No parameters	Locks the entire file (no record specification is given).
<i>record</i>	Locks the specified record number (for Random files) or byte (for Binary files).
to <i>end</i>	Locks from the beginning of the file to the specified record (for Random files) or byte (for Binary files).
<i>start</i> to <i>end</i>	Locks the specified range of records (for Random files) or bytes (for Binary files). The lock range must be the same as that used to subsequently unlock the file range, and all locked ranges must be unlocked before the file is closed. Ranges within files are not unlocked automatically by WM Basic when your script terminates, which can cause file access problems for other processes. It is a good idea to group the Lock and Unlock statements close together in the code, both for readability and so subsequent readers can see that the lock and unlock are performed on the same range. This practice also reduces errors in file locks.

Example 'This example creates test.dat and fills it with ten string variable 'records. These are displayed in a dialog box. The file is then reopened 'for read/write, and each record is locked, modified, rewritten, and 'unlocked. The new records are then displayed in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "This is record number: "
    b$ = "0"
    rec$ = ""

    msg = ""
    Open "test.dat" For Random Access Write Shared As #1
    For x = 1 To 10
        rec$ = a$ & x
        Lock #1,x
        Put #1,,rec$
        Unlock #1,x
        msg = msg & rec$ & crlf
    Next x
    Close
    MsgBox "The records are:" & crlf & msg

    msg = ""
    Open "test.dat" For Random Access Read Write Shared As #1
    For x = 1 To 10
        rec$ = Mid$(rec$,1,23) & (11 - x)
        Lock #1,x
        Put #1,x,rec$
        Unlock #1,x
        msg = msg & rec$ & crlf
    Next x
    MsgBox "The records are: " & crlf & msg
    Close

    Kill "test.dat"
End Sub
```

See Also Unlock (statement); Open (statement).

Platform(s) Windows and Macintosh.

Lof (function)

Syntax Lof (*filenumber*)

Description Returns a Long representing the number of bytes in the given file.

Comments The *filenumber* parameter is an Integer used by WM Basic to refer to the open file the number passed to the Open statement.

The file must currently be open.

Example 'This example creates a test file, writes ten records into it,
'then finds the length of the file and displays it in a message box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "This is record number: "

    Open "test.dat" For Random Access Write Shared As #1
    For x = 1 To 10
        rec$ = a$ & x
        put #1,,rec$
        msg = msg & rec$ & crlf
    Next x
    Close

    Open "test.dat" For Random Access Read Write Shared As #1
    r% = LoF(1)
    Close
    MsgBox "The length of test.dat is: " & r%
End Sub
```

See Also `Loc` (function); `Open` (statement); `FileLen` (function).

Platform(s) Windows and Macintosh.

Log (function)

Syntax `Log`(*number*)

Description Returns a `Double` representing the natural logarithm of a given number.

Comments The value of *number* must be a `Double` greater than 0.

The value of *e* is 2.71828.

Example 'This example calculates the natural log of 100 and displays it in
'a message box.

```
Sub Main()
    x# = Log(100)
    MsgBox "The natural logarithm of 100 is: " & x#
End Sub
```

See Also `Exp` (function).

Platform(s) Windows and Macintosh.

Long (data type)

Syntax `Long`

Description Long variables are used to hold numbers (with up to ten digits of precision) within the following range:

-2,147,483,648 <= Long <= 2,147,483,647

Internally, longs are 4-byte values. Thus, when appearing within a structure, longs require 4 bytes of storage. When used with binary or random files, 4 bytes of storage are required.

The type-declaration character for Long is &.

See Also Currency (data type); Date (data type); Double (data type); Integer (data type); Object (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); *DefType* (statement); CLng (function).

Platform(s) Windows and Macintosh.

LSet (statement)

Syntax 1 LSet *dest* = *source*

Syntax 2 LSet *dest_variable* = *source_variable*

Description Left-aligns the source string in the destination string or copies one user-defined type to another.

Comments Syntax 1

The LSet statement copies the source string *source* into the destination string *dest*. The *dest* parameter must be the name of either a String or Variant variable. The *source* parameter is any expression convertible to a string.

If *source* is shorter in length than *dest*, then the string is left-aligned within *dest*, and the remaining characters are padded with spaces. If *source* is longer in length than *dest*, then *source* is truncated, copying only the leftmost number of characters that will fit in *dest*.

The *destvariable* parameter specifies a String or Variant variable. If *destvariable* is a Variant containing Empty, then no characters are copied. If *destvariable* is not convertible to a String, then a runtime error occurs. A runtime error results if *destvariable* is Null.

Syntax 2

The source structure is copied byte for byte into the destination structure. This is useful for copying structures of different types. Only the number of bytes of the smaller of the two structures is copied. Neither the source structure nor the destination structure can contain strings.

Example 'This example replaces a 40-character string of asterisks (*) with 'an RSet and LSet string and then displays the result.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim msg, tmpstr$
    tmpstr$ = String$(40, "*")
    msg = "Here are two strings that have been right-" + crlf
    msg = msg & "and left-justified in a 40-character string."
    msg = msg & crlf & crlf
    RSet tmpstr$ = "Right->"
    msg = msg & tmpstr$ & crlf
    LSet tmpstr$ = "<-Left"
    msg = msg & tmpstr$ & crlf
    MsgBox msg
End Sub
```

See Also RSet (function).

Platform(s) Windows and Macintosh.

LTrim, LTrim\$ (functions)

Syntax LTrim[\$](*text*)

Description Returns *text* with the leading spaces removed.

Comments LTrim\$ returns a String, whereas LTrim returns a String variant.

Null is returned if *text* is Null.

Example 'This example displays a right-justified string and its LTrim result.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "               <= This is a right-justified string"
    b$ = LTrim$(a$)
    MsgBox a$ & crlf & b$
End Sub
```

See Also RTrim, RTrim\$ (functions); Trim, Trim\$ (functions).

Platform(s) Windows and Macintosh.

MacID (function)

Syntax MacID(*text*)

Description Returns a value representing a collection of same-type files on the Macintosh.

Comments Since this platform does not support wildcards (i.e., * or ?), this function is the only way to specify a group of files. This function can only be used with the following statements:

```
Kill Dir$ Shell AppActivate
```

The *text\$* parameter is a four-character string containing a file type, a resource type, an application signature, or an Apple event. A runtime error occurs if the MacID function is used on platforms other than the Macintosh.

Example 'This example retrieves the names of all the text files.

```
Sub Main()
  s$ = Dir$(MacID("TEXT"))      'Get the first text file.
  While s$ <> ""
    MsgBox s$                    'Display it.
    s$ = Dir$                    'Get the next text file in the list.
  Wend

  'Delete all the text files.
  Kill MacID("TEXT")
End Sub
```

See Also Kill (statement); Dir, Dir\$ (functions); Shell (statement); AppActivate (statement).

Platform(s) Macintosh.

MacScript (statement)

Syntax MacScript *script\$*

Description Executes the specified AppleScript script.

Comments When using the MacScript statement, you can separate multiple lines by embedding carriage returns:

```
MacScript "Beep" + Chr(13) + "Display Dialog ""Hello"""
```

If embedding carriage returns proves cumbersome, you can use the Inline statement. The following Inline statement is equivalent to the above example:

```
Inline MacScript
  Beep
  Display Dialog "Hello"
End Inline
```

Example

```
Sub Main()
  MacScript "display dialog ""AppleScript"""
End Sub
```

See Also Inline (statement).

Platform(s) Macintosh.

322 Working Model Basic User's Manual

Platform Requires Macintosh System 7.0 or later.

Notes:

Macintosh

Main (statement)

Syntax Sub Main()
End Sub

Description Defines the subroutine where execution begins.

Example Sub **Main**()
 MsgBox "This is the Main() subroutine and entry point."
End Sub

Platform(s) Windows and Macintosh.

Mci (function)

Syntax Mci(*command\$,result\$* [,*error\$*])

Description Executes an Mci command, returning an Integer indicating whether the command was successful.

Comments The Mci function takes the following parameters:

Parameter	Description
<i>command\$</i>	String containing the command to be executed.
<i>result\$</i>	String variable into which the result is placed. If the command doesn't return anything, then a zero-length string is returned. To ignore the returned string, pass a zero-length string: <div>r% = Mci("open chimes.wav type waveaudio","")</div>
<i>error\$</i>	Optional String variable into which an error string will be placed. A zero-length string will be returned if the function is successful.

Example 'This first example plays a wave file. The wave file is played to completion before execution can continue.

```
Sub Main()
    Dim result As String
    Dim ErrorMessage As String
    Dim Filename As String
    Dim rc As Integer

    'Establish name of file in the Windows directory.
    Filename = FileParse$(System.WindowsDirectory$ + "\" +
"chimes.wav")

    'Open the file and driver.
    rc = Mci("open " & Filename & " type waveaudio alias
CoolSound","",ErrorMessage)
    If (rc) Then
        'Error occurred--display error message to user.
        MsgBox ErrorMessage
        Exit Sub
    End If

    rc = Mci("play CoolSound wait","", "")      'Wait for sound to
finish.
    rc = Mci("close CoolSound","", "")          'Close driver and file.
End Sub
```

Example 'This next example shows how to query an Mci device and play an MIDI file in
'the background.

```
Sub Main()  
    Dim result As String  
    Dim ErrMsg As String  
    Dim Filename As String  
    Dim rc As Integer  
  
    'Check to see whether MIDI device can play for us.  
    rc = Mci("capability sequencer can play",result,ErrorMessage)  
  
    'Check for error.  
    If rc Then  
        MsgBox ErrorMessage  
        Exit Sub  
    End If  
  
    'Can it play?  
    If result <> "true" Then  
        MsgBox "MIDI device is not capable of playing."  
        Exit Sub  
    End If  
  
    'Assemble a filename from the Windows directory.  
    Filename = FileParse$(System.WindowsDirectory$ & "\" &  
"canyon.mid")  
  
    'Open the driver and file.  
    rc = Mci("open " & Filename & " type sequencer alias  
song",result$,ErrMsg)  
    If rc Then  
        MsgBox ErrMsg  
        Exit Sub  
    End If  
  
    rc = Mci("play song","","")      'Play in the background.  
    MsgBox "Press OK to stop the music.",vbOKOnly  
    rc = Mci("close song","","")  
End Sub
```

See Also Beep (statement).

Platform(s) Windows.

Platform The Mci function accepts any Mci command as defined in the *Multimedia*

Notes: *Programmers Reference* in the Windows 3.1 SDK.
Windows

Menu (statement)

Syntax Menu *MenuItem\$*

Description Issues the specified menu command from the active window of the active application.

Comments The *MenuItem\$* parameter specifies the complete menu item name, with each menu level being separated by a period. For example, the "Open" command on the "File" menu is represented by "File.Open". Cascading menu items may have multiple periods, one for each pop-up menu, such as "File.Layout.Vertical". Menu items can also be specified using numeric index values. For example, to select the third menu item from the File menu, use "File.#3". To select the fourth item from the third menu, use "#3.#4".

Items from an application's system menu can be selected by beginning the menu item specification with a period, such as ".Restore" or ".Minimize".

A runtime error will result if the menu item specification does not specify a menu item. For example, "File" specifies a menu pop-up rather than a menu item, and "File.Blah Blah" is not a valid menu item.

When comparing menu item names, this statement removes periods (.), spaces, and the ampersand. Furthermore, all characters after a backspace or tab are removed. Thus, the menu item "&Open...\aCtrl+F12" translates simply to "Open".

A runtime error is generated if the menu item cannot be found or is not enabled at the time that this statement is encountered.

Examples

```
Sub Main()
    Menu "File.Open"
    Menu "Format.Character.Bold"
    Menu ".Restore"           'Command from system menu
    Menu "File.#2"
End Sub
```

See Also MenuItemChecked (function); MenuItemEnabled (function); MenuItemExists (function).

Platform(s) Windows.

MenuItemChecked (function)

Syntax MenuItemChecked (*MenuItemName\$*)

Description Returns True if the given menu item exists and is checked; returns False otherwise.

Comments The *MenuItemName\$* parameter specifies a complete menu item or menu item pop-up following the same format as that used by the Menu statement.

Example 'This example turns the ruler off if it is on.

```
Sub Main()  
    If MenuItemChecked("View.Ruler") Then Menu "View.Ruler"  
End Sub
```

See Also Menu (statement); MenuItemEnabled (function); MenuItemExists (function).

Platform(s) Windows.

MenuItemEnabled (function)

Syntax MenuItemEnabled(*MenuItemName\$*)

Description Returns True if the given menu item exists and is enabled; returns False otherwise.

Comments The *MenuItemName\$* parameter specifies a complete menu item or menu item pop-up following the same format as that used by the Menu statement.

Example 'This example only pastes if there is something in the Clipboard.

```
Sub Main()  
    If MenuItemEnabled("Edit.Paste") Then  
        Menu "Edit.Paste"  
    Else  
        MsgBox "There is nothing in the Clipboard.",vbOKOnly  
    End If  
End Sub
```

See Also Menu (statement); MenuItemChecked (function); MenuItemExists (function).

Platform(s) Windows.

MenuItemExists (function)

Syntax MenuItemExists(*MenuItemName\$*)

Description Returns True if the given menu item exists; returns False otherwise.

Comments The *MenuItemName\$* parameter specifies a complete menu item or menu item pop-up following the same format as that used by the Menu statement.

Examples

```
Sub Main()  
    If MenuItemExists("File.Open") Then Beep  
    If MenuItemExists("File") Then MsgBox "There is a File menu."  
End Sub
```

See Also Menu (statement); MenuItemChecked (function); MenuItemEnabled (function).

Platform(s) Windows.

Mid, Mid\$ (functions)

Syntax `Mid[$](text, start [, length])`

Description Returns a substring of the specified string, beginning with *start*, for *length* characters.

Comments The returned substring starts at character position *start* and will be *length* characters long.

Mid\$ returns a String, whereas Mid returns a String variant.

The Mid/Mid\$ functions take the following parameters:

Parameter	Description
<i>text</i>	Any String expression containing the text from which characters are returned.
<i>start</i>	Integer specifying the character position where the substring begins. If <i>start</i> is greater than the length of <i>text</i> \$, then a zero-length string is returned.
<i>length</i>	Integer specifying the number of characters to return. If this parameter is omitted, then the entire string is returned, starting at <i>start</i> .

The Mid function will return Null if *text* is Null.

Example 'This example displays a substring from the middle of a string variable 'using the Mid\$ function and replaces the first four characters with "NEW " using the Mid\$ statement.

```
Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()
    a$ = "This is the Main string containing text."
    b$ = Mid$(a$,13,Len(a$))
    Mid$(b$,1) = "NEW "
    MsgBox a$ & crlf & b$
End Sub
```

See Also InStr (function); Option Compare (statement); Mid, Mid\$ (statements).

Platform(s) Windows and Macintosh.

Mid, Mid\$ (statements)

Syntax `Mid[$](variable, start[, length]) = newvalue`

Description Replaces one part of a string with another.

Comments The Mid/Mid\$ statements take the following parameters:

Parameter	Description
<i>variable</i>	String or Variant variable to be changed.
<i>start</i>	Integer specifying the character position within <i>variable</i> where replacement begins. If <i>start</i> is greater than the length of <i>variable</i> , then <i>variable</i> remains unchanged.
<i>length</i>	Integer specifying the number of characters to change. If this parameter is omitted, then the entire string is changed, starting at <i>start</i> .
<i>newvalue</i>	Expression used as the replacement. This expression must be convertible to a String.

The resultant string is never longer than the original length of *variable*.

With Mid, *variable* must be a Variant variable convertible to a String, and *newvalue* is any expression convertible to a string. A runtime error is generated if either variant is Null.

Example 'This example displays a substring from the middle of a string
'variable using the Mid\$ function, replacing the first four characters
'with "NEW " using the Mid\$ statement.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a$ = "This is the Main string containing text."
    b$ = Mid$(a$,13,Len(a$))
    Mid$(b$,1) = "NEW "
    MsgBox a$ & crlf & b$
End Sub
```

See Also Mid, Mid\$ (functions); Option Compare (statement).

Platform(s) Windows and Macintosh.

Minute (function)

Syntax Minute(*time*)

Description Returns the minute of the day encoded in the specified *time* parameter.

Comments The value returned is as an Integer between 0 and 59 inclusive.

The *time* parameter is any expression that converts to a Date.

Example 'This example takes the current time; extracts the hour, minute, 'and second; and displays them as the current time.

```
Sub Main()  
    xt# = TimeValue(Time$())  
    xh# = Hour(xt#)  
    xm# = Minute(xt#)  
    xs# = Second(xt#)  
    MsgBox "The current time is: " & xh# & ":" & xm# & ":" & xs#  
End Sub
```

See Also Day (function); Second (function); Month (function); Year (function); Hour (function); Weekday (function); DatePart (function).

Platform(s) Windows and Macintosh.

MIRR (function)

Syntax MIRR(*ValueArray*(), *FinanceRate*, *ReinvestRate*)

Description Returns a Double representing the modified internal rate of return for a series of periodic payments and receipts.

Comments The modified internal rate of return is the equivalent rate of return on an investment in which payments and receipts are financed at different rates. The interest cost of investment and the rate of interest received on the returns on investment are both factors in the calculations.

The MIRR function requires the following parameters:

Parameter	Description
<i>ValueArray</i> ()	<p>Array of Double numbers representing the payments and receipts. Positive values are payments (invested capital), and negative values are receipts (returns on investment).</p> <p>There must be at least one positive (investment) value and one negative (return) value.</p>
<i>FinanceRate</i>	Double representing the interest rate paid on invested monies (paid out).
<i>ReinvestRate</i>	<p>Double representing the rate of interest received on incomes from the investment (receipts).</p> <p><i>FinanceRate</i> and <i>ReinvestRate</i> should be expressed as percentages. For example, 11 percent should be expressed as 0.11.</p> <p>To return the correct value, be sure to order your payments and receipts in the correct sequence.</p>

Example 'This example illustrates the purchase of a lemonade stand for \$800
'financed with money borrowed at 10%. The returns are estimated to
'accelerate as the stand gains popularity. The proceeds are placed
'in a bank at 9 percent interest. The incomes are estimated (generated)
'over 12 months. This program first generates the income stream array
'in two For...Next loops, and then the modified internal rate of return
is
'calculated and displayed. Notice that the annual rates are normalized
'to monthly rates by dividing them by 12.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim valu#(12)
    valu(1) = -800                                'Initial investment
    msg = valu(1) & ", "
    For x = 2 To 5
        valu(x) = 100 + (x * 2)                    'Incomes months 2-5
        msg = msg & valu(x) & ", "
    Next x
    For x = 6 To 12
        valu(x) = 100 + (x * 10)                    'Incomes months 6-12
        msg = msg & valu(x) & ", "
    Next x
    retn# = MIRR(valu,.1/12,.09/12)                'Note: normalized annual
rates
                                                rates

    msg = "The values: " & crlf & msg & crlf & crlf
    MsgBox msg & "Modified rate: " & Format(retn#,"Percent")
End Sub
```

See Also Fv (function); IRR (function); Npv (function); Pv (function).

Platform(s) Windows and Macintosh.

MkDir (statement)

Syntax MkDir *dir\$*

Description Creates a new directory as specified by *dir\$*.

Example 'This example creates a new directory on the default drive. If
'this causes an error, then the error is displayed and the program
'terminates. If no error is generated, the directory is removed with
'the Rmdir statement.

```
Sub Main()
    On Error Resume Next
    MkDir "TestDir"
    If Err <> 0 Then
        MsgBox "The following error occurred: " & Error(Err)
    Else
        MsgBox "Directory was created and is about to be removed."
        Rmdir "TestDir"
    End If
End Sub
```

See Also ChDir (statement); ChDrive (statement); CurDir, CurDir\$ (functions); Dir, Dir\$ (functions); Rmdir (statement).

Platform(s) Windows and Macintosh.

Platform This command behaves the same as the DOS "mkdir" command.

Notes:
Windows

Mod (operator)

Syntax *expression1* Mod *expression2*

Description Returns the remainder of *expression1* / *expression2* as a whole number.

Comments If both expressions are integers, then the result is an integer. Otherwise, each expression is converted to a Long before performing the operation, returning a Long.

A runtime error occurs if the result overflows the range of a Long.

If either expression is Null, then Null is returned. Empty is treated as 0.

Example 'This example uses the Mod operator to determine the value of a randomly
'selected card where card 1 is the ace (1) of clubs and card 52 is the
'king (13) of spades. Since the values recur in a sequence of 13 cards
'within 4 suits, we can use the Mod function to determine the value of
'any given card number.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    cval$ =
"ACE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,TEN,JACK,QUEEN,KING"
    Randomize
    card% = Random(1,52)
    value = card% Mod 13
    If value = 0 Then value = 13
    CardNum$ = Item$(cval,value)
    If card% < 53 Then suit$ = "spades"
    If card% < 40 Then suit$ = "hearts"
    If card% < 27 Then suit$ = "diamonds"
    If card% < 14 Then suit$ = "clubs"
    msg = "Card number " & card% & " is the "
    msg = msg & CardNum & " of " & suit$
    MsgBox msg
End Sub
```

See Also / (operator); \ (operator).

Platform(s) Windows and Macintosh.

Month (function)

Syntax `Month(date)`

Description Returns the month of the date encoded in the specified *date* parameter.

Comments The value returned is as an `Integer` between 1 and 12 inclusive.

The *date* parameter is any expression that converts to a `Date`.

Example 'This example returns the current month in a dialog box.

```
Sub Main()  
    mons$ = "Jan., Feb., Mar., Apr., May, Jun., Jul., Aug., Sep., Oct.,  
Nov., Dec."  
    tdate$ = Date$  
    tmonth! = Month(DateValue(tdate$))  
    MsgBox "The current month is: " & Item$(mons$,tmonth!)  
End Sub
```

See Also `Day (function)`; `Minute (function)`; `Second (function)`; `Year (function)`; `Hour (function)`; `Weekday (function)`; `DatePart (function)`.

Platform(s) Windows and Macintosh.

MsgBox (function)

Syntax `MsgBox(msg [, [type] [, title]])`

Description Displays a message in a dialog box with a set of predefined buttons, returning an `Integer` representing which button was selected.





Comments The `MsgBox` function takes the following parameters:

Parameter	Description
<i>msg</i>	<p>Message to be displayed—any expression convertible to a <code>String</code>.</p> <p>End-of-lines can be used to separate lines (either a carriage return, line feed, or both). If a given line is too long, it will be word-wrapped. If <i>msg</i> contains character 0, then only the characters up to the character 0 will be displayed.</p> <p>The width and height of the dialog box are sized to hold the entire contents of <i>msg</i>.</p> <p>A runtime error is generated if <i>msg</i> is <code>Null</code>.</p>
<i>type</i>	<p><code>Integer</code> specifying the type of dialog box (see below).</p>
<i>title</i>	<p>Caption of the dialog box. This parameter is any expression convertible to a <code>String</code>. If it is omitted, then <code>BasicScript</code> is used.</p> <p>A runtime error is generated if <i>title</i> is <code>Null</code>.</p>

The MsgBox function returns one of the following values:

Constant	Value	Description
vbOK	1	OK was clicked.
vbCancel	2	Cancel was clicked.
vbAbort	3	Abort was clicked.
vbRetry	4	Retry was clicked.
vbIgnore	5	Ignore was clicked.
vbYes	6	Yes was clicked.
vbNo	7	No was clicked.

The *type* parameter is the sub of any of the following values:

Constant	Value	Description
ebOKOnly	0	Displays OK button only.
ebOKCancel	1	Displays OK and Cancel buttons.
ebAbortRetryIgnore	2	Displays Abort, Retry, and Ignore buttons.
ebYesNoCancel	3	Displays Yes, No, and Cancel buttons.
ebYesNo	4	Displays Yes and No buttons.
ebRetryCancel	5	Displays Retry and Cancel buttons.
ebCritical	16	Displays "stop" icon. 
ebQuestion	32	Displays "question mark" icon. 
ebExclamation	48	Displays "exclamation point" icon. 
ebInformation	64	Displays "information" icon. 
ebDefaultButton1	0	First button is the default button.
ebDefaultButton2	256	Second button is the default button.
ebDefaultButton3	512	Third button is the default button.
ebApplicationModal	0	Application modal—the current application is suspended until the dialog box is closed.
ebSystemModal	4096	System modal—all applications are suspended until the dialog box is closed.

The default value for *type* is 0 (display only the OK button, making it the default).

Breaking Text across Lines

The *msg* parameter can contain end-of-line characters, forcing the text that follows to start on a new line. The following example shows how to display a string on two lines:

```
MsgBox "This is on" + Chr(13) + Chr(10) + "two lines."
```

The carriage-return or line-feed characters can be used by themselves to designate an end-of-line.

```
r = MsgBox("Hello, World")
```



```
r = MsgBox("Hello, World",vbYesNoCancel Or vbDefaultButton1)
```



```
r = MsgBox("Hello, World",vbYesNoCancel Or vbDefaultButton1 Or  
vbCritical)
```



Example

```
Sub Main
    MsgBox "This is a simple message box."
    MsgBox "This is a message box with a title and an  
icon.",vbExclamation,"Simple"
    MsgBox "This message box has OK and Cancel  
buttons.",vbOkCancel,"MsgBox"
    MsgBox "This message box has Abort, Retry, and Ignore buttons.", _  
        vbAbortRetryIgnore,"MsgBox"
    MsgBox "This message box has Yes, No, and Cancel buttons.", _  
        vbYesNoCancel Or vbDefaultButton2,"MsgBox"
    MsgBox "This message box has Yes and No buttons.",vbYesNo,"MsgBox"
    MsgBox "This message box has Retry and Cancel  
buttons.",vbRetryCancel,"MsgBox"
    MsgBox "This message box is system modal!",vbSystemModal
End Sub
```

See Also AskBox\$ (function); AskPassword\$ (function); InputBox, InputBox\$ (functions); OpenFilename\$ (function); SaveFilename\$ (function); SelectBox (function); AnswerBox (function).

Platform(s) Windows and Macintosh.

Platform Notes: The appearance of the MsgBox dialog box and its icons differs slightly depending on the platform.

Platform Notes: MsgBox displays all text in its dialog box in 8-point MS Sans Serif.
Windows

MsgBox (statement)

Syntax MsgBox *msg* [, *type*] [, *title*]

Description This command is the same as the MsgBox function, except that the statement form does not return a value. See MsgBox (function).

Example

```
Sub Main()  
    MsgBox "This is text displayed in a message box." 'Display text.  
    MsgBox "The result is: " & (10 * 45) 'Display a number.  
End Sub
```

See Also AskBox\$ (function); AskPassword\$ (function); InputBox, InputBox\$ (functions); OpenFilename\$ (function); SaveFilename\$ (function); SelectBox (function); AnswerBox (function).

Platform(s) Windows and Macintosh.

MsgClose (statement)

Syntax MsgClose

Description Closes the modeless message dialog box.

Comments Nothing will happen if there is no open message dialog box.

Example

```
Sub Main()  
    MsgOpen "Printing. Please wait...", 0, True, True  
    Sleep 3000  
    MsgClose  
End Sub
```

See Also MsgOpen (statement); MsgSetThermometer (statement); MsgSetText (statement).

Platform(s) Windows.

MsgOpen (statement)

Syntax `MsgOpen msg$,timeout,isCancel,isThermometer [,X,Y]`

Description This statement displays a message in a dialog box with an optional Cancel button and thermometer.

Comments The `MsgOpen` statement takes the following parameters:

Parameter	Description
<i>msg\$</i>	String containing the text to be displayed. The text can be changed using the <code>MsgSetText</code> statement.
<i>timeout</i>	Integer specifying the number of seconds before the dialog box is automatically removed. The <i>timeout</i> parameter has no effect if its value is 0.
<i>isCancel</i>	Boolean controlling whether or not a Cancel button appears within the dialog box beneath the displayed message. If this parameter is <code>True</code> , then a Cancel button appears. If it is not specified or <code>False</code> , then no Cancel button is created. If a user chooses the Cancel button at runtime, a trappable runtime error is generated (error number 809). In this manner, a message dialog box can be displayed and processing can continue as normal, aborting only when the user cancels the process by choosing the Cancel button.
<i>isThermometer</i>	Boolean controlling whether the dialog box contains a thermometer. If this parameter is <code>True</code> , then a thermometer is created between the text and the optional Cancel button. The thermometer initially indicates 0% complete and can be changed using the <code>MsgSetThermometer</code> statement.
<i>X, Y</i>	Integer coordinates specifying the location of the upper left corner of the message box, in twips (twentieths of a point). If these parameters are not specified, then the window is centered on top of the application. Unlike other dialog boxes, a message dialog box remains open until the user selects Cancel, the timeout has expired, or the <code>MsgClose</code> statement is executed (this is sometimes referred to as modeless). Only a single message window can be opened at any one time. The message window is removed automatically when a script terminates.

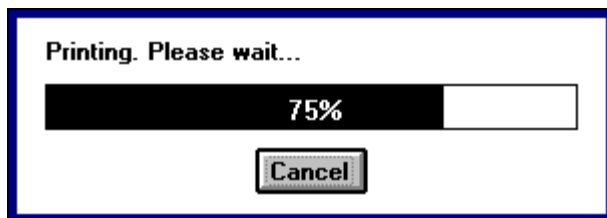
```
MsgOpen "Printing. Please wait...",0,False,False
```



```
MsgOpen "Printing. Please wait...",0,True,False
```



```
MsgOpen "Printing. Please wait...",0,True,True  
MsgSetThermometer 75
```



Example 'The following example displays several types of message boxes.

```
Sub Main()  
  MsgOpen "Printing. Please wait...",0,True,False  
  Sleep 3000  
  MsgClose  
  MsgOpen "Printing. Please wait...",0,True,True  
  For x = 1 to 100  
    MsgSetThermometer x  
  Next x  
  Sleep 1000  
  Msgclose  
End Sub
```

See Also `MsgClose (statement)`; `MsgSetThermometer (statement)`; `MsgSetText (statement)`.

Platform(s) Windows.

MsgSetText (statement)

Syntax `MsgSetText newtext$`

Description Changes the text within an open message dialog box (one that was previously opened with the `MsgOpen` statement).

Comments The message dialog box is not resized to accommodate the new text.
A runtime error will result if a message dialog box is not currently open (using MsgOpen).

Example 'This example creates a modeless message box, leaving room in the message
'text for the record number. This box contains a Cancel button.

```
Sub Main()  
    MsgOpen "Reading Record",0,True,False  
    For i = 1 To 100  
        'Read a record here.  
        'Update the modeless message box.  
        Sleep 100  
        MsgSetText "Reading record " & i  
    Next i  
    MsgClose  
End Sub
```

See Also MsgClose (statement); MsgOpen (statement); MsgSetThermometer (statement).

Platform(s) Windows.

MsgSetThermometer (statement)

Syntax `MsgSetThermometer percentage`

Description Changes the percentage filled indicated within the thermometer of a message dialog box (one that was previously opened with the MsgOpen statement).

Comments A runtime error will result if a message box is not currently open (using MsgOpen) or if the value of *percentage* is not between 0 and 100 inclusive.

Example 'This example create a modeless message box with a thermometer and a Cancel button. This example also shows how to process the clicking of the Cancel button.

```
Sub Main()  
    On Error Goto ErrorTrap  
    MsgOpen "Reading records from file...",0,True,True  
    For i = 1 To 100  
        'Read a record here.  
        'Update the modeless message box.  
        MsgSetThermometer i  
        DoEvents  
        Sleep 50  
    Next i  
    MsgClose  
    On Error Goto 0          'Turn error trap off.  
    Exit Sub  
  
ErrorTrap:  
    If Err = 809 Then  
        MsgBox "Cancel was pressed!"  
        Exit Sub          'Reset error handler.  
    End If  
End Sub
```

See Also MsgClose (statement); MsgOpen (statement); MsgSetText (statement).

Platform(s) Windows.

Name (statement)

Syntax Name *oldfile\$* As *newfile\$*

Description Renames a file.

Comments Each parameter must specify a single filename. Wildcard characters such as * and ? are not allowed.

Some platforms allow naming of files to different directories on the same physical disk volume. For example, the following rename will work under Windows:

```
Name "c:\samples\mydoc.txt" As "c:\backup\doc\mydoc.bak"
```

You cannot rename files across physical disk volumes. For example, the following will error under Windows:

```
Name "c:\samples\mydoc.txt" As "a:\mydoc.bak" 'This will error!
```

To rename a file to a different physical disk, you must first copy the file, then erase the original:

```
FileCopy "c:\samples\mydoc.txt", "a:\mydoc.bak" 'Make a copy
Kill "c:\samples\mydoc.txt" 'Delete the original
```

Example 'This example creates a file called test.dat and then renames it to test2.dat.

```
Sub Main()
  On Error Resume Next
  If FileExists("test.dat") Then
    Name "test.dat" As "test2.dat"
    If Err <> 0 Then
      msg = "File exists and cannot be renamed! Error: " & Err
    Else
      msg = "File exists and renamed to test2.dat."
    End If
  Else
    Open "test.dat" For Output As #1
    Close
    Name "test.dat" As "test2.dat"
    If Err <> 0 Then
      msg = "File created but not renamed! Error: " & Err
    Else
      msg = "File created and renamed to test2.dat."
    End If
  End If
  MsgBox msg
End Sub
```

See Also Kill (statement), FileCopy (statement).

Platform(s) Windows and Macintosh.

Net.AddCon (method)

Syntax Net.AddCon *netpath\$,password\$,localname\$*

Description Redirects a local device (a disk drive or printer queue) to the specified shared device or remote server.

Comments The `Net.AddCon` method takes the following parameters:

Parameter	Description
<i>netpath\$</i>	String containing the name of the shared device or the name of a remote server. This parameter can contain the name of a shared printer queue (such as that returned by <code>Net.Browse[1]</code>) or the name of a network path (such as that returned by <code>Net.Browse[0]</code>).
<i>password\$</i>	String containing the password for the given device or server. This parameter is mainly used to specify the password on a remote server.
<i>localname\$</i>	String containing the name of the local device being redirected, such as "LPT1" or "D:". A runtime error will result if no network is present.

Example 'This example sets N: so that it refers to the network path
SYS:\PUBLIC.

```
Sub Main()  
    Net.AddCon "SYS:\PUBLIC", "", "N:"  
End Sub
```

See Also `Net.CancelCon` (method); `Net.GetCon$` (method).

Platform(s) Windows.

Net.Browse\$ (method)

Syntax `Net.Browse$(type)`

Description Calls the currently installed network's browse dialog box, requesting a particular type of information.

Comments The *type* parameter is an Integer specifying the type of dialog box to display:

Type	Description
0	If <i>type</i> is 0, then this method displays a dialog box that allows the user to browse network volumes and directories. Choosing OK returns the completed pathname as a String.
1	If <i>type</i> is 1, then this function displays a dialog box that allows the user to browse the network's printer queues. Choosing OK returns the complete name of that printer queue as a String. This string is the same format as required by the <code>Net.AddCon</code> method. This dialog box differs depending on the type of network installed. A runtime error will result if no network is present.

Example 'This second example retrieves a valid network path.

```
Sub Main()
    s$ = Net.Browse$(0)
    If s$ <> "" Then
        MsgBox "The following network path was selected: " & s$
    Else
        MsgBox "Dialog box was canceled."
    End If
End Sub
```

See Also Net.Dialog (method).

Platform(s) Windows.

Net.CancelCon (method)

Syntax Net.CancelCon *connection\$* [, *isForce*]

Description Cancels a network connection.

Comments The Net.CancelCon method takes the following parameters:

Parameter	Description
<i>connection\$</i>	String containing the name of the device to cancel, such as "LPT1" or "D:".
<i>isForce</i>	Boolean specifying whether to force the cancellation of the connection if there are open files or open print jobs. If this parameter is True , then this method will close all open files and open print jobs before the connection is closed. If this parameter is False , this the method will issue a runtime error if there are any open files or open print jobs. A runtime error will result if no network is present.

Example 'This example deletes the drive mapping associated with drive N:.

```
Sub Main()
    Net.CancelCon "N:"
End Sub
```

See Also Net.AddCon (method); Net.GetCon\$ (method).

Platform(s) Windows.

Net.Dialog (method)

Syntax Net.Dialog

Description Displays the dialog box that allows configuration of the currently installed network.

Comments The displayed dialog box depends on the currently installed network. The dialog box is modal—script execution will be paused until the dialog box is completed.

A runtime error will result if no network is present.

Example 'This example invokes the network driver dialog box.

```
Sub Main()  
    Net.Dialog  
End Sub
```

See Also Net.Browse\$ (method).

Platform(s) Windows.

Net.GetCaps (method)

Syntax Net.GetCaps(*type*)

Description Returns an Integer specifying information about the network and its capabilities.

Comments The *type* parameter specifies what type of information to retrieve:

Value of type	Description
1	Returns the version of the driver specification to which the currently installed network driver conforms. The high byte of the returned value contains the major version number and the low byte contains the minor version number. These values can be retrieved using the following code: <i>MajorVersionNumber</i> = Net.GetCaps(1) \ 256 <i>MinorVersionNumber</i> = Net.GetCaps(1) And &H00FF
2	Returns the type of network. The network type is returned in the high byte and the sub-network type is returned in the low byte. These values can be obtained using the following code: <i>NetType</i> = Net.GetCaps(2) \ 256 <i>SubNetType</i> = Net.GetCaps(2) And &H00FF

Using the above values, *NetType* can be any of the following values:

0	No network is installed.
1	Microsoft Network.
2	Microsoft LAN Manager.
3	Novell NetWare.
4	Banyan Vines.
5	10Net.
6	Locus.
7	SunSoft PC NFS.
8	LanStep.
9	9 Titles.
10	Articom Lantastic.
11	IBM AS/400.
12	FTP Software FTP NFS.
13	DEC Pathworks.

If *NetType* is 128, then *SubNetType* is any of the following values (you can test for any of these values using the `And` operator):

0	None.
bit &H0001	Microsoft Network.
bit &H0002	Microsoft LAN Manager.
bit &H0004	Windows for Workgroups.
bit &H0008	Novell NetWare.
bit &H0010	Banyan Vines.
bit &H0080	Other unspecified network.

3 Returns the network driver version number.

4 Returns 1 if the `Net.User$` property is supported, 0 otherwise.

6 Returns any of the following values indicating which connections are supported (you can test for these values using the `And` operator):

bit &H0001	Driver supports <code>Net.AddCon</code> .
bit &H0002	Driver supports <code>Net.CancelCon</code> .
bit &H0004	Driver supports <code>Net.GetCon</code> .
bit &H0008	Driver supports auto connect.
bit &H0010	Driver supports <code>Net.Browse\$</code> .

- 7 Returns a value indicating which printer function are available (you can test for these values using the `And` operator):
- | | |
|------------|---|
| bit &H0002 | Driver supports open print job. |
| bit &H0004 | Driver supports close print job. |
| bit &H0010 | Driver supports hold print job. |
| bit &H0020 | Driver supports release print job. |
| bit &H0040 | Driver supports cancel print job. |
| bit &H0080 | Driver supports setting the number of print copies. |
| bit &H0100 | Driver supports watch print queue. |
| bit &H0200 | Driver supports unwatch print queue. |
| bit &H0400 | Driver supports locking queue data. |
| bit &H0800 | Driver supports unlocking queue data. |
| bit &H1000 | Driver supports queue change message. |
| bit &H2000 | Driver supports abort print job. |
| bit &H4000 | Driver supports no arbitrary lock. |
| bit &H8000 | Driver supports write print job. |
- 8 Returns a value indicating which dialog functions are available (you can test for these values using the `And` operator):
- | | |
|------------|---|
| bit &H0001 | Driver supports device mode dialog. |
| bit &H0002 | Driver supports the Browse dialog. |
| bit &H0004 | Driver supports the Connect dialog. |
| bit &H0008 | Driver supports the Disconnect dialog. |
| bit &H0010 | Driver supports the View Queue dialog. |
| bit &H0020 | Driver supports the Property dialog. |
| bit &H0040 | Driver supports the Connection dialog. |
| bit &H0080 | Driver supports the Printer Connect dialog. |
| bit &H0100 | Driver supports the Shares dialog. |
| bit &H0200 | Driver supports the Share As dialog. |

A runtime error will result if no network is present.

Examples

```
Sub Main()  
    'This example checks the type of network.  
    If Net.GetCaps(2) = 768 Then MsgBox "This is a Novell network."  
  
    'This checks whether the net supports retrieval of the user name.  
    If Net.GetCaps(4) And 1 Then MsgBox "User name is: " + Net.User$  
  
    'This checks whether this net supports the Browse dialog boxes.  
    If Net.GetCaps(6) And &H0010 Then MsgBox Net.Browse$(1)  
End Sub
```

Platform(s) Windows.

Net.GetCon\$(method)

Syntax `Net.GetCon$(localname$)`

- Description** Returns the name of the network resource associated with the specified redirected local device.
- Comments** The *localname\$* parameter specifies the name of the local device, such as "LPT1" or "D:". The function returns a zero-length string if the specified local device is not redirected. A runtime error will result if no network is present.
- Example** 'This example finds out where drive Z is mapped.
- ```
Sub Main()
 NetPath$ = Net.GetCon$("Z:")
 MsgBox "Drive Z is mapped as " & NetPath$
End Sub
```
- See Also** Net.CancelCon (method); Net.AddCon (method).
- Platform(s)** Windows.

## Net.User\$ (property)

---

- Syntax** Net.User\$
- Description** Returns the name of the user on the network.
- Comments** A runtime error is generated if the network is not installed.
- Examples**
- ```
Sub Main()
    'This example tells the user who he or she is.
    MsgBox "You are " & Net.User$

    'This example makes sure this capability is supported.
    If Net.GetCaps(4) And 1 Then MsgBox "You are " & Net.User$
End Sub
```
- Platform(s)** Windows.

New (keyword)

- Syntax 1** Dim *ObjectVariable* As New *ObjectType*
- Syntax 2** Set *ObjectVariable* = New *ObjectType*
- Description** Creates a new instance of the specified object type, assigning it to the specified object variable.

Comments The `New` keyword is used to declare a new instance of the specified data object. This keyword can only be used with data object types.

At runtime, the application or extension that defines that object type is notified that a new object is being defined. The application responds by creating a new physical object (within the appropriate context) and returning a reference to that object, which is immediately assigned to the variable being declared.

When that variable goes out of scope (i.e., the `Sub` or `Function` procedure in which the variable is declared ends), the application is notified. The application then performs some appropriate action, such as destroying the physical object.

See Also `Dim (statement); Set (statement)`.

Platform(s) Windows and Macintosh.

Not (operator)

Syntax `Not expression`

Description Returns either a logical or binary negation of *expression*.

Comments The result is determined as shown in the following table:

If the Expression Is	Then the Result Is
True	False
False	True
Null	Null
Any numeric type	A binary negation of the number. If the number is an <code>Integer</code> , then an <code>Integer</code> is returned. Otherwise, the <i>expression</i> is first converted to a <code>Long</code> , then a binary negation is performed, returning a <code>Long</code> .
Empty	Treated as a <code>Long</code> value 0.

Example 'This example demonstrates the use of the Not operator in comparing 'logical expressions and for switching a True/False toggle variable.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    a = False
    b = True
    If (Not a and b) Then msg = "a = False, b = True" & crlf

    toggle% = True
    msg = msg & "toggle% is now " & Format(toggle%,"True/False") & crlf
    toggle% = Not toggle%
    msg = msg & "toggle% is now " & Format(toggle%,"True/False") & crlf
    toggle% = Not toggle%
    msg = msg & "toggle% is now " & Format(toggle%,"True/False")
    MsgBox msg
End Sub
```

See Also Boolean (data type); Comparison Operators (topic).

Platform(s) Windows and Macintosh.

Nothing (constant)

Description A value indicating that an object variable no longer references a valid object.

Example

```
Sub Main()
    Dim a As Object
    If a Is Nothing Then
        MsgBox "The object variable references no object."
    Else
        MsgBox "The object variable references: " & a.Value
    End If
End Sub
```

See Also Set (statement); Object (data type).

Platform(s) Windows and Macintosh.

Now (function)

Syntax Now[()]

Description Returns a Date variant representing the current date and time.

Example 'This example shows how the Now function can be used as an elapsed-time counter.

```
Sub Main()  
    t1# = Now()  
    MsgBox "Wait a while and click OK."  
    t2# = Now()  
    t3# = Second(t2#) - Second(t1#)  
    MsgBox "Elapsed time was: " & t3# & " seconds."  
End Sub
```

See Also Date, Date\$ (functions); Time, Time\$ (functions).

Platform(s) Windows and Macintosh.

NPer (function)

Syntax NPer (Rate,Pmt,Pv,Fv,Due)

Description Returns the number of periods for an annuity based on periodic fixed payments and a constant rate of interest.

Comments An annuity is a series of fixed payments paid to or received from an investment over a period of time. Examples of annuities are mortgages, retirement plans, monthly savings plans, and term loans.

The NPer function requires the following parameters:

Parameter	Description
<i>Rate</i>	Double representing the interest rate per period. If the periods are monthly, be sure to normalize annual rates by dividing them by 12.
<i>Pmt</i>	Double representing the amount of each payment or income. Income is represented by positive values, whereas payments are represented by negative values.
<i>Pv</i>	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, and the future value (see below) would be zero.
<i>Fv</i>	Double representing the future value of your annuity. In the case of a loan, the future value would be zero, and the present value would be the amount of the loan.
<i>Due</i>	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period. Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example 'This example calculates the number of \$100.00 monthly payments necessary
'to accumulate \$10,000.00 at an annual rate of 10%. Payments are made at the
'beginning of the month.

```
Sub Main()  
    ag# = NPer( (.10/12), 100, 0, 10000, 1)  
    MsgBox "The number of monthly periods is: " &  
    Format(ag#, "Standard")  
End Sub
```

See Also IPmt (function); Pmt (function); PPmt (function); Rate (function).

Platform(s) Windows and Macintosh.

Npv (function)

Syntax Npv(Rate, ValueArray())

Description Returns the net present value of an annuity based on periodic payments and receipts, and a discount rate.

Comments The Npv function requires the following parameters:

Parameter	Description
<i>Rate</i>	Double that represents the interest rate over the length of the period. If the values are monthly, annual rates must be divided by 12 to normalize them to monthly rates.
<i>ValueArray()</i>	<p>Array of Double numbers representing the payments and receipts. Positive values are payments, and negative values are receipts.</p> <p>There must be at least one positive and one negative value.</p> <p>Positive numbers represent cash received, whereas negative numbers represent cash paid out.</p> <p>For accurate results, be sure to enter your payments and receipts in the correct order because Npv uses the order of the array values to interpret the order of the payments and receipts.</p> <p>If your first cash flow occurs at the beginning of the first period, that value must be added to the return value of the Npv function. It should not be included in the array of cash flows.</p> <p>Npv differs from the Pv function in that the payments are due at the end of the period and the cash flows are variable. Pv's cash flows are constant, and payment may be made at either the beginning or end of the period.</p>

Example 'This example illustrates the purchase of a lemonade stand for \$800
'financed with money borrowed at 10%. The returns are estimated to
'accelerate as the stand gains popularity. The incomes are estimated
'(generated) over 12 months. This program first generates the income
'stream array in two For...Next loops, and then the net present value
'(Npv) is calculated and displayed. Note normalization of the annual
10%
'rate.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim valu#(12)
    valu(1) = -800                                'Initial investment
    msg = valu(1) & ", "
    For x = 2 To 5                                'Months 2-5
        valu(x) = 100 + (x * 2)
        msg = msg & valu(x) & ", "
    Next x
    For x = 6 To 12                                'Months 6-12
        valu(x) = 100 + (x * 10)                    'Accelerated income
        msg = msg & valu(x) & ", "
    Next x
    NetVal# = NPV((.10/12),valu)
    msg = "The values:" & crlf & msg & crlf & crlf
    MsgBox msg & "Net present value: " & Format(NetVal#,"Currency")
End Sub
```

See Also Fv (function); IRR (function); MIRR (function); Pv (function).

Platform(s) Windows and Macintosh.

Null (constant)

Description Represents a variant of VarType 1.

Comments The Null value has special meaning indicating that a variable contains no data.

Most numeric operators return Null when either of the arguments is Null. This "propagation" of Null makes it especially useful for returning error values through a complex expression. For example, you can write functions that return Null when an error occurs, then call this function within an expression. You can then use the IsNull function to test the final result to see whether an error occurred during calculation.

Since variants are Empty by default, the only way for Null to appear within a variant is for you to explicitly place it there. Only a few BasicScript functions return this value.

Example

```
Sub Main()  
    Dim a As Variant  
    a = Null  
    If IsNull(a) Then MsgBox "The variable is Null."  
    MsgBox "The VarType of a is: " & VarType(a)'Should display 1.  
End Sub
```

Platform(s) Windows and Macintosh.

Object (data type)

Syntax Object

Description A data type used to declare OLE automation variables.

Comments The Object type is used to declare variables that reference objects within an application using OLE automation.

Each object is a 4-byte (32-bit) value that references the object internally. The value 0 (or `Nothing`) indicates that the variable does not reference a valid object, as is the case when the object has not yet been given a value. Accessing properties or methods of such Object variables generates a runtime error.

Using Objects

Object variables are declared using the `Dim`, `Public`, or `Private` statement:

```
Dim MyApp As Object
```

Object variables can be assigned values (thereby referencing a real physical object) using the `Set` statement:

```
Set MyApp = CreateObject("phantom.application")  
Set MyApp = Nothing
```

Properties of an Object are accessed using the dot (.) separator:

```
MyApp.Color = 10  
i% = MyApp.Color
```

Methods of an Object are also accessed using the dot (.) separator:

```
MyApp.Open "sample.txt"  
isSuccess = MyApp.Save("new.txt",15)
```

Automatic Destruction

BasicScript keeps track of the number of variables that reference a given object so that the object can be destroyed when there are no longer any references to it:

```
Sub Main()                                'Number of references to object
  Dim a As Object                          '0
  Dim b As Object                          '0
  Set a = CreateObject("phantom.application") '1
  Set b = a                                '2
  Set a = Nothing                           '1
End Sub                                    '0 (object destroyed)
```

Note: An OLE automation object is instructed by BasicScript to destroy itself when no variables reference that object. However, it is the responsibility of the OLE automation server to destroy it. Some servers do not destroy their objects—usually when the objects have a visual component and can be destroyed manually by the user.

See Also Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Single (data type); String (data type); Variant (data type); Boolean (data type); *DefType* (statement).

Platform(s) Windows and Macintosh.

Objects (topic)

BasicScript defines two types of objects: data objects and OLE automation objects.

Syntactically, these are referenced in the same way.

What Is an Object

An object in BasicScript is an encapsulation of data and routines into a single unit. The use of objects in WM Basic has the effect of grouping together a set of functions and data items that apply only to a specific object type.

Objects expose data items for programmability called properties. For example, a `sheet` object may expose an integer called `NumColumns`. Usually, properties can be both retrieved (`get`) and modified (`set`).

Objects also expose internal routines for programmability called methods. In WM Basic, an object method can take the form of a function or a subroutine. For example, a OLE automation object called `MyApp` may contain a method subroutine called `Open` that takes a single argument (a filename), as shown below:

```
MyApp.Open "c:\files\sample.txt"
```

Declaring Object Variables

In order to gain access to an object, you must first declare an object variable using either `Dim`, `Public`, or `Private`:

```
Dim o As Object      'OLE automation object
```

Initially, objects are given the value 0 (or `Nothing`). Before an object can be accessed, it must be associated with a physical object.

Assigning a Value to an Object Variable

An object variable must reference a real physical object before accessing any properties or methods of that object. To instantiate an object, use the `Set` statement.

```
Dim MyApp As Object
Set MyApp = CreateObject("Server.Application")
```

Accessing Object Properties

Once an object variable has been declared and associated with a physical object, it can be modified using WM Basic code. Properties are syntactically accessible using the dot operator, which separates an object name from the property being accessed:

```
MyApp.BackgroundColor = 10
i% = MyApp.DocumentCount
```

Properties are set using WM Basic's normal assignment statement:

```
MyApp.BackgroundColor = 10
```

Object properties can be retrieved and used within expressions:

```
i% = MyApp.DocumentCount + 10
MsgBox "Number of documents = " & MyApp.DocumentCount
```

Accessing Object Methods

Like properties, methods are accessed via the dot operator. Object methods that do not return values behave like subroutines in WM Basic (i.e., the arguments are not enclosed within parentheses):

```
MyApp.Open "c:\files\sample.txt", True, 15
```

Object methods that return a value behave like function calls in WM Basic. Any arguments must be enclosed in parentheses:

```
If MyApp.DocumentCount = 0 Then MsgBox "No open documents."
NumDocs = app.count(4,5)
```

There is no syntactic difference between calling a method function and retrieving a property value, as shown below:

```
variable = object.property(arg1,arg2)
variable = object.method(arg1,arg2)
```

Comparing Object Variables

The values used to represent objects are meaningless to the script in which they are used, with the following exceptions:

- Objects can be compared to each other to determine whether they refer to the same object.
- Objects can be compared with `Nothing` to determine whether the object variable refers to a valid object.

Object comparisons are accomplished using the `Is` operator:

```
If a Is b Then MsgBox "a and b are the same object."  
If a Is Nothing Then MsgBox "a is not initialized."  
If b Is Not Nothing Then MsgBox "b is in use."
```

Collections

A collection is a set of related object variables. Each element in the set is called a member and is accessed via an index, either numeric or text, as shown below:

```
MyApp.Toolbar.Buttons(0)  
MyApp.Toolbar.Buttons("Tuesday")
```

It is typical for collection indexes to begin with 0.

Each element of a collection is itself an object, as shown in the following examples:

```
Dim MyToolbarButton As Object  
  
Set MyToolbarButton = MyApp.Toolbar.Buttons("Save")  
MyApp.Toolbar.Buttons(1).Caption = "Open"
```

The collection itself contains properties that provide you with information about the collection and methods that allow navigation within that collection:

```
Dim MyToolbarButton As Object  
  
NumButtons% = MyApp.Toolbar.Buttons.Count  
MyApp.Toolbar.Buttons.MoveNext  
MyApp.Toolbar.Buttons.FindNext "Save"  
  
For i = 1 To MyApp.Toolbar.Buttons.Count  
    Set MyToolbarButton = MyApp.Toolbar.Buttons(i)  
    MyToolbarButton.Caption = "Copy"  
Next i
```


Predefined Objects

WM Basic predefines a few objects for use in all scripts. These are:

Clipboard	System	Desktop	HWND
Net	Basic	Screen	

In addition, WM Basic has objects defined specifically for Working Model. Please see Chapter 1 and Chapter 3 for more information.

Note: Some of these objects are not available on all platforms.

Oct, Oct\$ (functions)

Syntax Oct[\$](*number*)

Description Returns a String containing the octal equivalent of the specified number.

Comments Oct\$ returns a String, whereas Oct returns a String variant.

The returned string contains only the number of octal digits necessary to represent the number.

The *number* parameter is any numeric expression. If this parameter is Null, then Null is returned. Empty is treated as 0. The *number* parameter is rounded to the nearest whole number before converting to the octal equivalent.

Example 'This example displays the octal equivalent of several numbers.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    st$ = "The octal values are: " & crlf
    For x = 1 To 5
        y% = x * 10
        st$ = st$ & y% & " : " & Oct$(y%) & crlf
    Next x
    MsgBox st$
End Sub
```

See Also Hex, Hex\$ (functions).

Platform(s) Windows and Macintosh.

OKButton (statement)

Syntax OKButton *X,Y,width,height* [, .Identifier]

Description Creates an OK button within a dialog box template.

Comments This statement can only appear within a dialog box template (i.e., between the `Begin Dialog` and `End Dialog` statements).

The `OKButton` statement accepts the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>.Identifier</i>	Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>).
	If the <i>DefaultButton</i> parameter is not specified in the <code>Dialog</code> statement, the OK button will be used as the default button. In this case, the OK button can be selected by pressing Enter on a nonbutton control.
	A dialog box template must contain at least one <code>OKButton</code> , <code>CancelButton</code> , or <code>PushButton</code> statement (otherwise, the dialog box cannot be dismissed).

Example 'This example shows how to use the OK and Cancel buttons within a dialog box template and how to detect which one closed the dialog box.

```
Sub Main()
    Begin Dialog ButtonTemplate 17,33,104,23,"Buttons"
        OKButton 8,4,40,14,.OK
        CancelButton 56,4,40,14,.Cancel
    End Dialog
    Dim ButtonDialog As ButtonTemplate
    WhichButton = Dialog(ButtonDialog)
    If WhichButton = -1 Then
        MsgBox "OK was pressed."
    ElseIf WhichButton = 0 Then
        MsgBox "Cancel was pressed."
    End If
End Sub
```

See Also `CancelButton (statement)`; `CheckBox (statement)`; `ComboBox (statement)`; `Dialog (function)`; `Dialog (statement)`; `DropListBox (statement)`; `GroupBox (statement)`; `ListBox (statement)`; `OptionButton (statement)`; `OptionGroup (statement)`; `Picture (statement)`; `PushButton (statement)`; `Text (statement)`; `TextBox (statement)`; `Begin Dialog (statement)`, `PictureButton (statement)`.

Platform(s) Windows and Macintosh.

On Error (statement)

Syntax `On Error {Goto label | Resume Next | Goto 0}`

Description Defines the action taken when a trappable runtime error occurs.

Comments The form `On Error Goto label` causes execution to transfer to the specified label when a runtime error occurs.

The form `On Error Resume Next` causes execution to continue on the line following the line that caused the error.

The form `On Error Goto 0` causes any existing error trap to be removed.

If an error trap is in effect when the script ends, then an error will be generated.

An error trap is only active within the subroutine or function in which it appears.

Once an error trap has gained control, appropriate action should be taken, and then control should be resumed using the `Resume` statement. The `Resume` statement resets the error handler and continues execution. If a procedure ends while an error is pending, then an error will be generated. (The `Exit Sub` or `Exit Function` statement also resets the error handler, allowing a procedure to end without displaying an error message.)

Errors within an Error Handler

If an error occurs within the error handler, then the error handler of the caller (or any procedure in the call stack) will be invoked. If there is no such error handler, then the error is fatal, causing the script to stop executing. The following statements reset the error state (i.e., these statements turn off the fact that an error occurred):

```
Resume
Err=-1
```

The `Resume` statement forces execution to continue either on the same line or on the line following the line that generated the error. The `Err=-1` statement allows explicit resetting of the error state so that the script can continue normal execution without resuming at the statement that caused the error condition.

The `On Error` statement will not reset the error. Thus, if an `On Error` statement occurs within an error handler, it has the effect of changing the location of a new error handler for any new errors that may occur once the error has been reset.

Example 'This example will demonstrate three types of error handling.
'The first case simply by-passes an expected error and continues
'with program operation. The second case creates an error branch
'that jumps to a common error handling routine that processes
'incoming errors, clears the error (with the Resume statement) and
'resumes program execution. The third case clears all internal error
'handling so that execution will stop when the next error is
'encountered.

```
Sub Main()  
    Dim x%  
    a = 10000  
    b = 10000  
  
    On Error Goto Pass          'Branch to this label on error.  
    Do  
        x% = a * b  
    Loop  
  
Pass:  
    Err = -1                    'Clear error status.  
    MsgBox "Cleared error status and continued."  
  
    On Error Goto Overflow      'Branch to new error routine on any  
    x% = 1000                   'subsequent errors.  
    x% = a * b  
    x% = a / 0  
  
    On Error Goto 0             'Clear error branching.  
    x% = a * b                  'Program will stop here.  
    Exit Sub                    'Exit before common error routine.  
  
Overflow:                       'Beginning of common error routine.  
    If Err = 6 then  
        MsgBox "Overflow Branch."  
    Else  
        MsgBox Error(Err)  
    End If  
  
    Resume Next  
End Sub
```

See Also Error Handling (topic); Error (statement); Resume (statement).

Platform(s) Windows and Macintosh.

Open (statement)

Syntax Open *filename*\$ [For *mode*] [Access *accessmode*] [*lock*] As [#] *filenumber* _
[Len = *reclen*]

Description Opens a file for a given mode, assigning the open file to the supplied *filenumber*.

Comments The *filename\$* parameter is a string expression that contains a valid filename.

The *filenumber* parameter is a number between 1 and 255. The `FreeFile` function can be used to determine an available file number.

The *mode* parameter determines the type of operations that can be performed on that file:

File Mode	Description
Input	Opens an existing file for sequential input (<i>filename\$</i> must exist). The value of <i>accessmode</i> , if specified, must be <code>Read</code> .
Output	Opens an existing file for sequential output, truncating its length to zero, or creates a new file. The value of <i>accessmode</i> , if specified, must be <code>Write</code> .
Append	Opens an existing file for sequential output, positioning the file pointer at the end of the file, or creates a new file. The value of <i>accessmode</i> , if specified, must be <code>Read Write</code> .
Binary	Opens an existing file for binary I/O or creates a new file. Existing binary files are never truncated in length. The value of <i>accessmode</i> , if specified, determines how the file can subsequently be accessed.
Random	Opens an existing file for record I/O or creates a new file. Existing random files are truncated only if <i>accessmode</i> is <code>Write</code> . The <i>reclen</i> parameter determines the record length for I/O operations.

If the *mode* parameter is missing, then `Random` is used.

The *accessmode* parameter determines what type of I/O operations can be performed on the file:

Access	Description
Read	Opens the file for reading only. This value is valid only for files opened in <code>Binary</code> , <code>Random</code> , or <code>Input</code> mode.
Write	Opens the file for writing only. This value is valid only for files opened in <code>Binary</code> , <code>Random</code> , or <code>Output</code> mode.
Read Write	Opens the file for both reading and writing. This value is valid only for files opened in <code>Binary</code> , <code>Random</code> , or <code>Append</code> mode.

If the *accessmode* parameter is not specified, the following defaults are used:

File Mode	Default Value for <i>accessmode</i>
Input	Read
Output	Write
Append	Read Write
Binary	When the file is initially opened, access is attempted three times in the following order: <div><div>1. Read Write</div><div>2. Write</div><div>3. Read</div></div>
Random	Same as Binary files

The *lock* parameter determines what access rights are granted to other processes that attempt to open the same file. The following table describes the values for *lock*:

<i>lock</i> Value	Description
Shared	Another process can both read this file and write to it. (Deny none.)
Lock Read	Another process can write to this file but not read it. (Deny read.)
Lock Write	Another process can read this file but not write to it. (Deny write.)
Lock Read Write	Another process is prevented both from reading this file and from writing to it. (Exclusive.)

If *lock* is not specified, then the file is opened in Shared mode.

If the file does not exist and the *lock* parameter is specified, the file is opened twice — once to create the file and again to establish the correct sharing mode.

Files opened in Random mode are divided up into a sequence of records, each of the length specified by the *reclen* parameter. If this parameter is missing, then 128 is used. For files opened for sequential I/O, the *reclen* parameter specifies the size of the internal buffer used by WM Basic when performing I/O. Larger buffers mean faster file access. For Binary files, the *reclen* parameter is ignored.

Example 'This example opens several files in various configurations.

```
Sub Main()
  Open "test.dat" For Output Access Write Lock Write As #2
  Close
  Open "test.dat" For Input Access Read Shared As #1
  Close
  Open "test.dat" For Append Access Write Lock Read Write as #3
  Close
  Open "test.dat" For Binary Access Read Write Shared As #4
  Close
  Open "test.dat" For Random Access Read Write Lock Read As #5
  Close
  Open "test.dat" For Input Access Read Shared As #6
  Close
  Kill "test.dat"
End Sub
```

See Also Close (statement); Reset (statement); FreeFile (function).

Platform(s) Windows and Macintosh.

OpenFilename\$ (function)

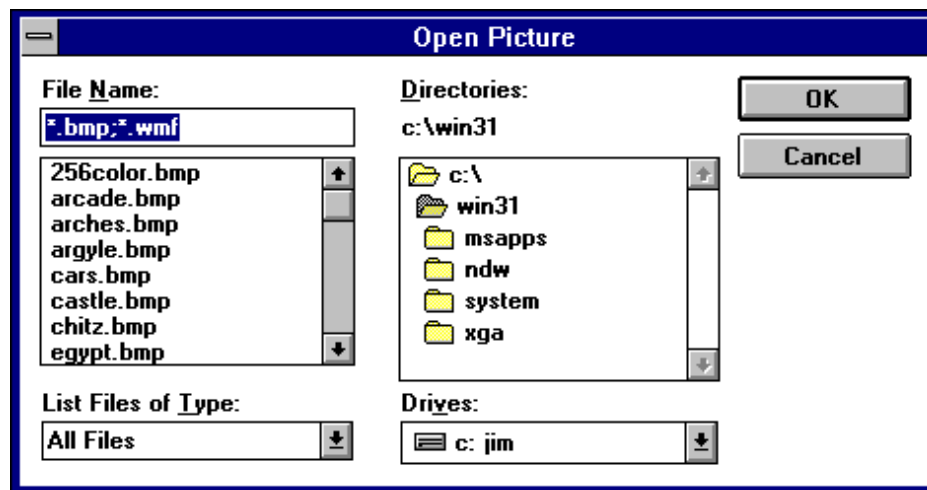
Syntax OpenFilename\$([*title\$* [, *extensions\$*]])

Description Displays a dialog box that prompts the user to select from a list of files, returning the full pathname of the file the user selects or a zero-length string if the user selects Cancel.

Comments This function displays the standard file open dialog box, which allows the user to select a file. It takes the following parameters:

Parameter	Description
<i>title\$</i>	String specifying the title that appears in the dialog box's title bar. If this parameter is omitted, then "Open" is used.
<i>extension\$</i>	String specifying the available file types. The format for this string depends on the platform on which WM Basic is running. If this parameter is omitted, then all files are displayed.

```
e$ = "All Files:*.BMP;*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF"
f$ = OpenFilename$("Open Picture",e$)
```



Example 'This example asks the user for the name of a file, then proceeds to read the first line from that file.

```
Sub Main
  Dim f As String,s As String
  f$ = OpenFilename$("Open Picture","Text Files:*.TXT")
  If f$ <> "" Then
    Open f$ For Input As #1
    Line Input #1,s$
    Close #1
    MsgBox "First line from " & f$ & " is " & s$
  End If
End Sub
```

See Also MsgBox (statement); AskBox\$ (function); AskPassword\$ (function); InputBox, InputBox\$ (functions); SaveFilename\$ (function); SelectBox (function); AnswerBox (function).

Platform(s) Windows and Macintosh.

Platform Notes: Windows Placeholder	Description
<i>type</i>	Specifies the name of the grouping of files, such as All Files.
<i>ext</i>	Specifies a valid file extension, such as *.BAT or *.*F? For example, the following are valid <i>extensions\$</i> specifications: <pre>"All Files:*.*" "Documents:*.TXT,*.DOC" "All Files:*.*;Documents:*.TXT,*.DOC"</pre>
Platform Notes: Macintosh	On the Macintosh, the <i>extensions\$</i> parameter contains a comma-separated list of four-character file types. For example: <pre>"TEXT,XLS4,MSWD"</pre> On the Macintosh, the <i>title\$</i> parameter is ignored.

Operator Precedence (topic)

The following table shows the precedence of the operators supported by WM Basic. Operations involving operators of higher precedence occur before operations involving operators of lower precedence. When operators of equal precedence occur together, they are evaluated from left to right.

Operator	Description	Precedence Order
()	Parentheses	Highest
^	Exponentiation	
-	Unary minus	
/, *	Division and multiplication	
\	Integer division	
Mod	Modulo	
+, -	Addition and subtraction	
&	String concatenation	
=, <>, >, <, <=, >=	Relational	
Like, Is	String and object comparison	
Not	Logical negation	
And	Logical or binary conjunction	
Or	Logical or binary disjunction	
Xor, Eqv, Imp	Logical or binary operators	Lowest

The precedence order can be controlled using parentheses, as shown below:

```
a = 4 + 3 * 2      'a becomes 10.
a = (4 + 3) * 2    'a becomes 14.
```

Operator Precision (topic)

When numeric, binary, logical or comparison operators are used, the data type of the result is generally the same as the data type of the more precise operand. For example, adding an `Integer` and a `Long` first converts the `Integer` operand to a `Long`, then performs a long addition, overflowing only if the result cannot be contained with a `Long`. The order of precision is shown in the following table:

Empty	Least precise
Boolean	
Integer	
Long	
Single	
Date	
Double	
Currency	Most precise

There are exceptions noted in the descriptions of each operator.

The rules for operand conversion are further complicated when an operator is used with variant data. In many cases, an overflow causes automatic promotion of the result to the next highest precise data type. For example, adding two `Integer` variants results in an `Integer` variant unless it overflows, in which case the result is automatically promoted to a `Long` variant.

Option Base (statement)

Syntax `Option Base {0 | 1}`

Description Sets the lower bound for array declarations.

Comments By default, the lower bound used for all array declarations is 0. This statement must appear outside of any functions or subroutines.

Example `Option Base 1`

```
Sub Main()
    Dim a(10)           'Contains 10 elements (not 11).
End Sub
```

See Also `Dim (statement); Public (statement); Private (statement).`

Platform(s) Windows and Macintosh.

Option Compare (statement)

Syntax `Option Compare [Binary | Text]`

Description Controls how strings are compared.

Comments When `Option Compare` is set to `Binary`, then string comparisons are case-sensitive (e.g., "A" does not equal "a"). When it is set to `Text`, string comparisons are case-insensitive (e.g., "A" is equal to "a").

The default value for `Option Compare` is `Binary`.

The `Option Compare` statement affects all string comparisons in any statements that follow the `Option Compare` statement. Additionally, the setting affects the default behavior of `Instr`, `StrComp`, and the `Like` operator. The following table shows the types of string comparisons affected by this setting:

>	<	<>
<=	>=	<code>Instr</code>
<code>StrComp</code>	<code>Like</code>	

The `Option Compare` statement must appear outside the scope of all subroutines and functions. In other words, it cannot appear within a `Sub` or `Function` block.

Example 'This example shows the use of Option Compare.

```
Option Compare Binary
Sub CompareBinary
    a$ = "This String Contains UPPERCASE."
    b$ = "this string contains uppercase."
    If a$ = b$ Then
        MsgBox "The two strings were compared case-insensitive."
    Else
        MsgBox "The two strings were compared case-sensitive."
    End If
End Sub

Option Compare Text
Sub CompareText
    a$ = "This String Contains UPPERCASE."
    b$ = "this string contains uppercase."
    If a$ = b$ Then
        MsgBox "The two strings were compared case-insensitive."
    Else
        MsgBox "The two strings were compared case-sensitive."
    End If
End Sub

Sub Main()
    CompareBinary           'Calls subroutine above.
    CompareText             'Calls subroutine above.
End Sub
```

See Also Like (operator); InStr (function); StrComp (function); Comparison Operators (topic).

Platform(s) Windows and Macintosh.

Option CStrings (statement)

Syntax Option CStrings {On | Off}

Description Turns on or off the ability to use C-style escape sequences within strings.

Comments When `Option CStrings On` is in effect, the compiler treats the backslash character as an escape character when it appears within strings. An escape character is simply a special character that cannot otherwise be ordinarily typed by the computer keyboard.

Escape	Description	Equivalent Expression
<code>\r</code>	Carriage return	<code>Chr\$(13)</code>
<code>\n</code>	Line feed	<code>Chr\$(10)</code>
<code>\a</code>	Bell	<code>Chr\$(7)</code>
<code>\b</code>	Backspace	<code>Chr\$(8)</code>
<code>\f</code>	Form feed	<code>Chr\$(12)</code>
<code>\t</code>	Tab	<code>Chr\$(9)</code>
<code>\v</code>	Vertical tab	<code>Chr\$(11)</code>
<code>\0</code>	Null	<code>Chr\$(0)</code>
<code>\"</code>	Double quotation mark	<code>" "</code> or <code>Chr\$(34)</code>
<code>\\</code>	Backslash	<code>Chr\$(92)</code>
<code>\?</code>	Question mark	<code>?</code>
<code>\'</code>	Single quotation mark	<code>'</code>
<code>\xhh</code>	Hexadecimal number	<code>Chr\$(Val("&Hhh"))</code>
<code>\ooo</code>	Octal number	<code>Chr\$(Val("&Oooo"))</code>
<code>\anycharacter</code>	Any character	<i>anycharacter</i>

With hexadecimal values, WM Basic stops scanning for digits when it encounters a nonhexadecimal digit or two digits, whichever comes first. Similarly, with octal values, WM Basic stops scanning when it encounters a nonoctal digit or three digits, whichever comes first.

When `Option CStrings Off` is in effect, then the backslash character has no special meaning. This is the default.

Example `Option CStrings On`

```
Sub Main()  
    MsgBox "They said, \"Watch out for that clump of grass!\""  
    MsgBox "First line.\r\nSecond line."  
    MsgBox "Char A: \x41 \r\n Char B: \x42"  
End Sub
```

Platform(s) Windows and Macintosh.

OptionButton (statement)

Syntax `OptionButton X,Y,width,height,title$ [, .Identifier]`

Description Defines an option button within a dialog box template.

Comments This statement can only appear within a dialog box template (i.e., between the `Begin Dialog` and `End Dialog` statements).

The `OptionButton` statement accepts the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>title\$</i>	String containing text that appears within the option button. This text may contain an ampersand character to denote an accelerator letter, such as "&Portrait" for Portrait, which can be selected by pressing the P accelerator.
<i>.Identifier</i>	Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>).

Example See `OptionGroup` (statement).

See Also `CancelButton` (statement); `CheckBox` (statement); `ComboBox` (statement); `Dialog` (function); `Dialog` (statement); `DropListBox` (statement); `GroupBox` (statement); `ListBox` (statement); `OKButton` (statement); `OptionGroup` (statement); `Picture` (statement); `PushButton` (statement); `Text` (statement); `TextBox` (statement); `Begin Dialog` (statement), `PictureButton` (statement).

Platform(s) Windows and Macintosh.

Platform Notes: On Windows, accelerators are underlined, and the accelerator combination `Alt+letter` is used.

Windows

Platform Notes: On the Macintosh, accelerators are normal in appearance, and the accelerator combination `Command+letter` is used.

Macintosh

OptionEnabled (function)

Syntax `OptionEnabled(name$ | id)`

Description Returns `True` if the specified option button is enabled within the current window or dialog box; returns `False` otherwise.

Comments This function is used to determine whether a given option button is enabled within the current window or dialog box. If an option button is enabled, then its value can be set using the `SetOption` statement.

The `OptionEnabled` statement takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the option button.
<i>id</i>	Integer specifying the ID of the option button.
Note: The <code>OptionEnabled</code> function is used to determine whether an option button is enabled in another application's dialog box. Use the <code>DlgEnable</code> function with dynamic dialog boxes.	

Example 'This example checks to see whether the option button is enabled before setting it.

```
If OptionEnabled("Tile") Then
    SetOption "Tile"
End If
```

See Also `GetOption` (function); `OptionExists` (function); `SetOption` (statement).

Platform(s) Windows.

OptionExists (function)

Syntax `OptionExists(name$ | id)`

Description Returns `True` if the specified option button exists within the current window or dialog box; returns `False` otherwise.

Comments This function is used to determine whether a given option button exists within the current window or dialog box.

The `OptionExists` statement takes the following parameters:

Parameter	Description
<i>name\$</i>	String containing the name of the option button.
<i>id</i>	Integer specifying the ID of the option button.
Note: The <code>OptionExists</code> function is used to determine whether an option button exists in another application's dialog box. There is no equivalent function for use with dynamic dialog boxes.	

Example 'This example checks to see whether the option button exists and is enabled
'before setting it.

```
If OptionExists("Tile") Then
    If OptionEnabled("Tile") Then
        SetOption("Tile")
    End If
End If
```

See Also GetOption (function); OptionEnabled (function); SetOption (statement).

Platform(s) Windows.

OptionGroup (statement)

Syntax OptionGroup *.Identifier*

Description Specifies the start of a group of option buttons within a dialog box template.

Comments The *.Identifier* parameter specifies the name by which the group of option buttons can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the selected option button within the group (0 is the first option button, 1 is the second option button, and so on). This variable can be accessed using the following syntax:
DialogVariable.Identifier.

This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).

When the dialog box is created, the option button specified by *.Identifier* will be on; all other option buttons in the group will be off. When the dialog box is dismissed, the *.Identifier* will contain the selected option button.

Example 'This example creates a group of option buttons.

```
Sub Main()
    Begin Dialog PrintTemplate 16,31,128,65,"Print"
        GroupBox 8,8,64,52,"Orientation",.Junk
        OptionGroup .Orientation
            OptionButton 16,20,37,8,"Portrait",.Portrait
            OptionButton 16,32,51,8,"Landscape",.Landscape
            OptionButton 16,44,49,8,"Don't Care",.DontCare
        OKButton 80,8,40,14
    End Dialog
    Dim PrintDialog As PrintTemplate
    Dialog PrintDialog
End Sub
```

See Also CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); Picture (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

Platform(s) Windows and Macintosh.

Or (operator)

Syntax *expression1 Or expression2*

Description Performs a logical or binary disjunction on two expressions.

Comments If both expressions are either Boolean, Boolean variants, or Null variants, then a logical disjunction is performed as follows:

If the first expression is	and the second expression is	then the result is
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

Binary Disjunction

If the two expressions are Integer, then a binary disjunction is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary disjunction is then performed, returning a Long result.

Binary disjunction forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:

1	Or	1	=	1	Example:
0	Or	1	=	1	5 10101001
1	Or	0	=	1	6 01101010
0	Or	0	=	0	Or 11101011

Examples 'This first example shows the use of logical Or.

```
Dim s$ As String

s$ = InputBox$("Enter a string.")
If s$ = "" Or Mid$(s$,1,1) = "A" Then
    s$ = LCase$(s$)
End If

'This second example shows the use of binary Or.
```

```
Dim w As Integer

TryAgain:
    s$ = InputBox$("Enter a hex number (four digits max).")
    If Mid$(s$,1,1) <> "&" Then
        s$ = "&H" & s$
    End If
    If Not IsNumeric(s$) Then Goto TryAgain

    w = CInt(s$)
    MsgBox "Your number is &H" & Hex$(w)
    w = w Or &H8000
    MsgBox "Your number with the high bit set is &H" & Hex$(w)
```

See Also Operator Precedence (topic); Xor (operator); Eqv (operator); Imp (operator); And (operator).

Platform(s) Windows and Macintosh.

Pi (constant)

Syntax Pi

Description The Double value 3.141592653589793238462643383279.

Comments Pi can also be determined using the following formula:

$$4 * \text{Atn}(1)$$

Example 'This example illustrates the use of the Pi constant.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    dia# = 5
    circ# = Pi * dia#
    area# = Pi * ((dia# / 2) ^ 2)
    msg = "Diameter: 5" & crlf
    msg = msg & "Circumference: " & Format(circ#,"Standard") & crlf
    msg = msg & "Area: " & Format(area#,"Standard")
    MsgBox msg
End Sub
```

See Also Tan (function); Atn (function); Cos (function); Sin (function).

Platform(s) Windows and Macintosh.

Picture (statement)

Syntax `Picture X,Y,width,height,PictureName$,PictureType [, [.Identifier] [,style]]`

Description Creates a picture control in a dialog box template.

Comments Picture controls are used for the display of graphics images only. The user cannot interact with these controls.

The `Picture` statement accepts the following parameters:

Parameter	Description				
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.				
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.				
<i>PictureName\$</i>	String containing the name of the picture. If <i>PictureType</i> is 0, then this name specifies the name of the file containing the image. If <i>PictureType</i> is 10, then <i>PictureName\$</i> specifies the name of the image within the resource of the picture library. If <i>PictureName\$</i> is empty, then no picture will be associated with the control. A picture can later be placed into the picture control using the <code>DlgSetPicture</code> statement.				
<i>PictureType</i>	Integer specifying the source for the image. The following sources are supported: <table> <tr> <td>0</td><td>The image is contained in a file on disk.</td></tr> <tr> <td>10</td><td>The image is contained in a picture library as specified by the <code>PicName\$</code> parameter on the <code>Begin Dialog</code> statement.</td></tr> </table>	0	The image is contained in a file on disk.	10	The image is contained in a picture library as specified by the <code>PicName\$</code> parameter on the <code>Begin Dialog</code> statement.
0	The image is contained in a file on disk.				
10	The image is contained in a picture library as specified by the <code>PicName\$</code> parameter on the <code>Begin Dialog</code> statement.				
<i>.Identifier</i>	Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>). If omitted, then the first two words of <i>PictureName\$</i> are used.				
<i>style</i>	Specifies whether the picture is drawn within a 3D frame. It can be any of the following values: <table> <tr> <td>0</td><td>Draw the picture control with a normal frame.</td></tr> <tr> <td>1</td><td>Draw the picture control with a 3D frame.</td></tr> </table> If omitted, then the picture control is drawn with a normal frame.	0	Draw the picture control with a normal frame.	1	Draw the picture control with a 3D frame.
0	Draw the picture control with a normal frame.				
1	Draw the picture control with a 3D frame.				

The picture control extracts the actual image from either a disk file or a picture library. In the case of bitmaps, both 2- and 16-color bitmaps are supported. In the case of WMFs, WM Basic supports the Placeable Windows Metafile.

If *PictureName\$* is a zero-length string, then the picture is removed from the picture control, freeing any memory associated with that picture.

Examples

'This first example shows how to use a picture from a file.

```
Sub Main()
    Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction"
        OKButton 240,8,40,14
        Picture 8,8,224,64,"c:\bitmaps\logo.bmp",0,.Logo
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

'This second example shows how to use a picture from a picture library with
'a 3D frame.

```
Sub Main()
    Begin Dialog LogoDialogTemplate
16,31,288,76,"Introduction",,"pictures.dll"
        OKButton 240,8,40,14
        Picture 8,8,224,64,"CompanyLogo",10,.Logo,1
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

See Also CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropDownListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement); PictureButton (statement); DlgSetPicture (statement).

Platform(s) Windows and Macintosh.

Platform Picture controls can contain either a bitmap or a WMF (Windows metafile).
Notes: When extracting images from a picture library, WM Basic assumes that the
Windows resource type for metafiles is 256.

Picture libraries are implemented as DLLs on Windows.

Platform Picture controls on the Macintosh can contain only PICT images. These are
Notes: contained in files of type PICT.

Macintosh Picture libraries on the Macintosh are files with collections of named PICT resources. The *PictureName\$* parameter corresponds to the name of one the resources as it appears within the file.

PictureButton (statement)

Syntax `PictureButton X,Y,width,height,PictureName$,PictureType [,.Identifier]`

Description Creates a picture button control in a dialog box template.

Comments Picture button controls behave very much like a push button controls. Visually, picture buttons are different than push buttons in that they contain a graphic image imported either from a file or from a picture library.

The `PictureButton` statement accepts the following parameters:

Parameter	Description				
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.				
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.				
<i>PictureName\$</i>	String containing the name of the picture. If <i>PictureType</i> is 0, then this name specifies the name of the file containing the image. If <i>PictureType</i> is 10, then <i>PictureName\$</i> specifies the name of the image within the resource of the picture library. If <i>PictureName\$</i> is empty, then no picture will be associated with the control. A picture can later be placed into the picture control using the <code>DlgSetPicture</code> statement.				
<i>PictureType</i>	Integer specifying the source for the image. The following sources are supported: <table> <tr> <td>0</td><td>The image is contained in a file on disk.</td></tr> <tr> <td>10</td><td>The image is contained in a picture library as specified by the <i>PicName\$</i> parameter on the <code>Begin Dialog</code> statement.</td></tr> </table>	0	The image is contained in a file on disk.	10	The image is contained in a picture library as specified by the <i>PicName\$</i> parameter on the <code>Begin Dialog</code> statement.
0	The image is contained in a file on disk.				
10	The image is contained in a picture library as specified by the <i>PicName\$</i> parameter on the <code>Begin Dialog</code> statement.				
<i>.Identifier</i>	Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>). The picture button control extracts the actual image from either a disk file or a picture library, depending on the value of <i>PictureType</i> . The supported picture formats vary from platform to platform. If <i>PictureName\$</i> is a zero-length string, then the picture is removed from the picture button control, freeing any memory associated with that picture.				

Examples

'This first example shows how to use a picture from a file.

```
Sub Main()
    Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction"
        OKButton 240,8,40,14
        PictureButton 8,4,224,64,"c:\bitmaps\logo.bmp",0,.Logo
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

'This second example shows how to use a picture from a picture library.

```
Sub Main()
    Begin Dialog LogoDialogTemplate
16,31,288,76,"Introduction",,,"pictures.dll"
        OKButton 240,8,40,14
        PictureButton 8,4,224,64,"CompanyLogo",10,.Logo
    End Dialog
    Dim LogoDialog As LogoDialogTemplate
    Dialog LogoDialog
End Sub
```

See Also

CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); PushButton (statement); Text (statement); TextBox (statement); Begin Dialog (statement), Picture (statement), DlgSetPicture (statement).

Platform(s) Windows and Macintosh.

Platform Notes: Picture controls can contain either a bitmap or a WMF (Windows metafile). When extracting images from a picture library, WM Basic assumes that the resource type for metafiles is 256.

Windows Picture libraries are implemented as DLLs on Windows.

Platform Notes: Picture controls on the Macintosh can contain only PICT images. These are contained in files of type PICT.

Macintosh Picture libraries on the Macintosh are files with collections of named PICT resources. The *PictureName\$* parameter corresponds to the name of one the resources as it appears within the file.

Pmt (function)

Syntax `Pmt (Rate, NPer, Pv, Fv, Due)`

Description Returns the payment for an annuity based on periodic fixed payments and a constant rate of interest.

Comments An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The `Pmt` function requires the following parameters:

Parameter	Description
<i>Rate</i>	Double representing the interest rate per period. If the periods are given in months, be sure to normalize annual rates by dividing them by 12.
<i>NPer</i>	Double representing the total number of payments in the annuity.
<i>Pv</i>	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
<i>Fv</i>	Double representing the future value of your annuity. In the case of a loan, the future value would be 0.
<i>Due</i>	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.

Rate and *NPer* must be expressed in the same units. If *Rate* is expressed in months, then *NPer* must also be expressed in months.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example

```
'This example calculates the payment necessary to repay a $1,000.00
loan
'over 36 months at an annual rate of 10%. Payments are due at the
beginning
'of the period.

Sub Main()
    x = Pmt((.1/12),36,1000.00,0,1)
    msg = "The payment to amortize $1,000 over 36 months @ 10% is: "
    MsgBox msg & Format(x,"Currency")
End Sub
```

See Also `IPmt` (function); `NPer` (function); `PPmt` (function); `Rate` (function).

Platform(s) Windows and Macintosh.

PopupMenu (function)

Syntax `PopupMenu (MenuItems$ ())`

Description Displays a pop-up menu containing the specified items, returning an Integer representing the index of the selected item.

Comments If no item is selected (i.e., the pop-up menu is canceled), then a value of 1 less than the lower bound is returned (normally, -1).

This function creates a pop-up menu using the string elements in the given array. Each array element is used as a menu item. A zero-length string results in a separator bar in the menu.

The pop-up menu is created with the upper left corner at the current mouse position.

A runtime error results if *MenuItems\$* is not a single-dimension array.

Only one pop-up menu can be displayed at a time. An error will result if another script executes this function while a pop-up menu is visible.

Example

```
Sub Main()
    Dim a$()
    AppList a$
    w% = PopupMenu(a$)
End Sub
```



See Also `SelectBox` (function).

Platform(s) Windows.

PPmt (function)

Syntax `PPmt (Rate, Per, NPer, Pv, Fv, Due)`

Description Calculates the principal payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Comments An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The PPmt function requires the following parameters:

Parameter	Description
<i>Rate</i>	Double representing the interest rate per period.
<i>Per</i>	Double representing the number of payment periods. <i>Per</i> can be no less than 1 and no greater than <i>NPer</i> .
<i>NPer</i>	Double representing the total number of payments in your annuity.
<i>Pv</i>	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
<i>Fv</i>	Double representing the future value of your annuity. In the case of a loan, the future value would be 0.
<i>Due</i>	Integer indicating when payments are due. If this parameter is 0, then payments are due at the end of each period; if it is 1, then payments are due at the start of each period.

Rate and *NPer* must be in the same units to calculate correctly. If *Rate* is expressed in months, then *NPer* must also be expressed in months.

Negative values represent payments paid out, whereas positive values represent payments received.

Example 'This example calculates the principal paid during each year on a loan of
 of
 '\$1,000.00 with an annual rate of 10% for a period of 10 years. The
 result
 'is displayed as a table containing the following information: payment,
 'principal payment, principal balance.

```

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    pay = Pmt(.1,10,1000.00,0,1)
    msg = "Amortization table for 1,000" & crlf & "at 10% annually for"
    msg = msg & " 10 years: " & crlf & crlf
    bal = 1000.00
    For per = 1 to 10
        prn = PPmt(.1,per,10,1000,0,0)
        bal = bal + prn
        msg = msg & Format(pay,"Currency") & "    " &
Format$(Prn,"Currency")
        msg = msg & "    " & Format(bal,"Currency") & crlf
    Next per
    MsgBox msg
End Sub

```

See Also IPmt (function); NPer (function); Pmt (function); Rate (function).

Platform(s) Windows and Macintosh.

Print (statement)

Syntax Print [[{Spc(*n*) | Tab(*n*)}][*expressionlist*][{; | ,}]]

Description Prints data to an output device.

Comments The actual output device depends on the platform on which WM Basic is running.

The following table describes how data of different types is written:

Data Type	Description
String	Printed in its literal form, with no enclosing quotes.
Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.
Boolean	Printed as "True" or "False".
Date	Printed using the short date format. If either the date or time component is missing, only the provided portion is printed (this is consistent with the "general date" format understood by the <code>Format/Format\$</code> functions).
Empty	Nothing is printed.
Null	Prints "Null".

User-defined errors Printed as "Error *code*", where *code* is the value of the user-defined error. The word "Error" is not translated.

Each expression in *expressionlist* is separated with either a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces.

If the last expression in the list is not followed by a comma or a semicolon, then a carriage return is printed to the file. If the last expression ends with a semicolon, no carriage return is printed the next `Print` statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.

The `Tab` and `SpC` functions provide additional control over the column position. The `Tab` function moves the file position to the specified column, whereas the `SpC` function outputs the specified number of spaces.

Examples

```

Sub Main()
    i% = 10
    s$ = "This is a test."
    Print "The value of i=";i%,"the value of s=";s$

    'This example prints the value of i% in print zone 1 and s$ in
print
    'zone 3.
    Print i%,,s$

    'This example prints the value of i% and s$ separated by 10 spaces.
    Print i%;Spc(10);s$

    'This example prints the value of i in column 1 and s$ in column
30.
    Print i%;Tab(30);s$

    'This example prints the value of i% and s$.
    Print i%;s$,
    Print 67
End Sub

```

See Also ViewportOpen (statement).

Platform(s) Windows and Macintosh.

Platform Under Windows, this statement writes data to a viewport window.

Notes: If no viewport window is open, then the statement is ignored. Printing
Windows information to a viewport window is a convenient way to output debugging information. To open a viewport window, use the following statement:

ViewportOpen

Platform On the Macintosh, the Print statement prints data to stdout.

Notes:
Macintosh

Print# (statement)

Syntax Print [#]filename, [[{Spc(n) | Tab(n)}][expressionlist][{:|,}]]

Description Writes data to a sequential disk file.

Comments The *filenumber* parameter is a number that is used by WM Basic to refer to the open file—the number passed to the `Open` statement.

The following table describes how data of different types is written:

Data Type	Description
String	Printed in its literal form, with no enclosing quotes.
Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.
Boolean	Printed as "True" or "False".
Date	Printed using the short date format. If either the date or time component is missing, only the provided portion is printed (this is consistent with the "general date" format understood by the <code>Format/Format\$</code> functions).
Empty	Nothing is printed.
Null	Prints "Null".

User-defined errors Printed to files as "Error *code*", where *code* is the value of the user-defined error. The word "Error" is not translated.

Each expression in *expressionlist* is separated with either a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces.

If the last expression in the list is not followed by a comma or a semicolon, then an end-of-line is printed to the file. If the last expression ends with a semicolon, no end-of-line is printed—the next `Print` statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.

The `Write` statement always outputs information ending with an end-of-line. Thus, if a `Print` statement is followed by a `Write` statement, the file pointer is positioned on a new line.

The `Print` statement can only be used with files that are opened in `Output` or `Append` mode.

The `Tab` and `Spc` functions provide additional control over the file position. The `Tab` function moves the file position to the specified column, whereas the `Spc` function outputs the specified number of spaces.

In order to correctly read the data using the `Input#` statement, you should write the data using the `Write` statement.

Examples

```

Sub Main()
    'This example opens a file and prints some data.
    Open "test.dat" For Output As #1
    i% = 10
    s$ = "This is a test."
    Print #1,"The value of i=";i%,"the value of s=";s$

    'This example prints the value of i% in print zone 1 and s$ in
    'print zone 3.
    Print #1,i%,,s$

    'This example prints the value of i% and s$ separated by ten
    spaces.
    Print #1,i%;SpC(10);s$

    'This example prints the value of i in column 1 and s$ in column
    30.
    Print #1,i%;Tab(30);s$

    'This example prints the value of i% and s$.
    Print #1,i%;s$,
    Print #1,67

    Close #1
    Kill "test.dat"
End Sub

```

See Also Open (statement); Put (statement); Write# (statement).

Platform(s) Windows and Macintosh.

Platform Notes: The end-of-line character is different on many platforms. On some platforms, it is defined as a carriage-return/line-feed pair, and on other platforms, it is defined as only a line feed. The WM Basic statements that read sequential files do not care about the end-of-line character—either will work.

PrinterGetOrientation (function)

Syntax PrinterGetOrientation[()]

Description Returns an Integer representing the current orientation of paper in the default printer.

Comments PrinterGetOrientation returns ebPortrait if the printer orientation is set to portrait; otherwise, it returns ebLandscape.

This function loads the printer driver and therefore may be slow.

Example 'This example toggles the printer orientation.

```
Sub Main()  
  If PrinterGetOrientation = ebLandscape Then  
    PrinterSetOrientation ebPortrait  
  Else  
    PrinterSetOrientation ebLandscape  
  End If  
End Sub
```

See Also PrinterSetOrientation (statement).

Platform(s) Windows.

Platform The default printer is determined by examining the device= line in the

Notes: [windows] section of the win.ini file.

Windows

PrinterSetOrientation (statement)

Syntax PrinterSetOrientation *NewSetting*

Description Sets the orientation of the default printer to *NewSetting*.

Comments The possible values for *NewSetting* are as follows

Setting	Description
ebLandscape	Sets printer orientation to landscape.
ebPortrait	Sets printer orientation to portrait.
	This function loads the printer driver for the default printer and therefore may be slow.

Example See PrinterGetOrientation (function).

See Also PrinterGetOrientation (function).

Platform(s) Windows.

Platform The default printer is determined by examining the device= line in the

Notes: [windows] section of the win.ini file.

Windows

PrintFile (function)

Syntax PrintFile(*filename*\$)

Description Prints the *filename*\$ using the application to which the file belongs.

Comments PrintFile returns an Integer indicating success or failure.

If an error occurs executing the associated application, then PrintFile generates a trappable runtime error, returning 0 for the result. Otherwise, PrintFile returns a value representing that application to the system. This value is suitable for calling the AppActivate statement.

Example 'This example asks the user for the name of a text file, then prints it.

```
Sub Main()
    f$ = OpenFilename$("Print Text File","Text Files:*.txt")
    If f$ <> "" Then
        rc% = PrintFile(f$)
        If rc% > 32 Then
            MsgBox "File is printing."
        End If
    End If
End Sub
```

See Also Shell (function).

Platform(s) Windows.

Platform Notes: This function invokes the Windows 3.1 shell functions that cause an application to execute and print a file. The application executed by PrintFile depends on your system's file associations.

Windows

Private (statement)

Syntax Private *name* [(*subscripts*)] [As *type*] [, *name* [(*subscripts*)] [As *type*]]...

Description Declares a list of private variables and their corresponding types and sizes.

Comments Private variables are global to every Sub and Function within the currently executing script.

If a type-declaration character is used when specifying *name* (such as %, @, &, \$, or !), the optional [As *type*] expression is not allowed. For example, the following are allowed:

```
Private foo As Integer
Private foo%
```

The *subscripts* parameter allows the declaration of arrays. This parameter uses the following syntax:

```
[lower To] upper [, [lower To] upper]...
```

The *lower* and *upper* parameters are integers specifying the lower and upper bounds of the array. If *lower* is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Up to 60 array dimensions are allowed.

The total size of an array (not counting space for strings) is limited to 64K.

Dynamic arrays are declared by not specifying any bounds:

```
Private a()
```

The *type* parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Single, Double, Currency, Object, data object, built-in data type, or any user-defined data type.

If a variable is seen that has not been explicitly declared with either Dim, Public, or Private, then it will be implicitly declared local to the routine in which it is used.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the String type-declaration character:

```
Private name As String * length
```

where *length* is a literal number specifying the string's length.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value
------------------	----------------------

Integer	0
Long	0
Double	0.0
Single	0.0
Currency	0.0
Object	Nothing
Date	December 31, 1899 00:00:00
Boolean	False
Variant	Empty
String	" " (zero-length string)

User-defined type Each element of the structure is given a default value, as described above.

Arrays Each element of the array is given a default value, as described above.

Example See `Public (statement)`.

See Also `Dim (statement)`; `Redim (statement)`; `Public (statement)`; `Option Base (statement)`.

Platform(s) Windows and Macintosh.

Public (statement)

Syntax `Public name [(subscripts)] [As type] [, name [(subscripts)] [As type]]...`

Description Declares a list of public variables and their corresponding types and sizes.

Comments Public variables are global to all Subs and Functions in all scripts.

If a type-declaration character is used when specifying *name* (such as %, @, &, \$, or !), the optional [As *type*] expression is not allowed. For example, the following are allowed:

```
Public foo As integer
Public foo%
```

The *subscripts* parameter allows the declaration of arrays. This parameter uses the following syntax:

```
[lower To] upper [, [lower To] upper]...
```

The *lower* and *upper* parameters are integers specifying the lower and upper bounds of the array. If *lower* is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Up to 60 array dimensions are allowed.

The total size of an array (not counting space for strings) is limited to 64K.

Dynamic arrays are declared by not specifying any bounds:

```
Public a()
```

The *type* parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Single, Double, Currency, Object, data object, built-in data type, or any user-defined data type.

If a variable is seen that has not been explicitly declared with either Dim, Public, or Private, then it will be implicitly declared local to the routine in which it is used.

For compatibility, the keyword Global is also supported. It has the same meaning as Public.

Fixed-Length Strings

Fixed-length strings are declared by adding a length to the String type-declaration character:

```
Public name As String * length
```

where *length* is a literal number specifying the string's length.

Initial Values

All declared variables are given initial values, as described in the following table:

Data Type	Initial Value
Integer	0
Long	0
Double	0.0
Single	0.0
Currency	0.0
Date	December 31, 1899 00:00:00
Object	Nothing
Boolean	False
Variant	Empty
String	"" (zero-length string)

User-defined type Each element of the structure is given a default value, as described above.

Arrays Each element of the array is given a default value, as described above.

Sharing Variables

When sharing variables, you must ensure that the declarations of the shared variables are the same in each script that uses those variables. If the public variable being shared is a user-defined structure, then the structure definitions must be exactly the same.

Example 'This example uses a subroutine to calculate the area of ten circles
'and displays the result in a dialog box. The variables R and Ar are
'declared as Public variables so that they can be used in both Main and
Area.

```
Const crlf = Chr$(13) + Chr$(10)

Public x#, ar#

Sub Area()
    ar# = (x# ^ 2) * Pi
End Sub

Sub Main()
    msg = "The area of the ten circles are:" & crlf
    For x# = 1 To 10
        Area
        msg = msg & x# & ": " & ar# & Basic.Eoln$
    Next x#
    MsgBox msg
End Sub
```

See Also Dim (statement); Redim (statement); Private (statement); Option Base (statement).

Platform(s) Windows and Macintosh.

PushButton (statement)

Syntax PushButton *X,Y,width,height,title\$* [, *.Identifier*]

Description Defines a push button within a dialog box template.

Comments Choosing a push button causes the dialog box to close (unless the dialog function redefines this behavior).

This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements).

The PushButton statement accepts the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates specifying the position of the control (in dialog units) relative to the upper left corner of the dialog box.
<i>width, height</i>	Integer coordinates specifying the dimensions of the control in dialog units.
<i>title\$</i>	String containing the text that appears within the push button. This text may contain an ampersand character to denote an accelerator letter, such as "&Save" for Save.
<i>.Identifier</i>	<p>Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).</p> <p>If a push button is the default button, it can be selected by pressing Enter on a nonbutton control.</p> <p>A dialog box template must contain at least one OKButton, CancelButton, or PushButton statement (otherwise, the dialog box cannot be dismissed).</p>

Example 'This example creates a bunch of push buttons and displays
'which button was pushed.

```
Sub Main()
    Begin Dialog ButtonTemplate 17,33,104,84,"Buttons"
        OKButton 8,4,40,14,.OK
        CancelButton 8,24,40,14,.Cancel
        PushButton 8,44,40,14,"1",.Button1
        PushButton 8,64,40,14,"2",.Button2
        PushButton 56,4,40,14,"3",.Button3
        PushButton 56,24,40,14,"4",.Button4
        PushButton 56,44,40,14,"5",.Button5
        PushButton 56,64,40,14,"6",.Button6
    End Dialog
    Dim ButtonDialog As ButtonTemplate
    WhichButton% = Dialog(ButtonDialog)
    MsgBox "You pushed button " & WhichButton%
End Sub
```

See Also CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); Text (statement); TextBox (statement); Begin Dialog (statement); PictureButton (statement).

Platform(s) Windows and Macintosh.

Platform Notes: On Windows, accelerators are underlined, and the accelerator combination Alt+*letter* is used.

Windows

Platform Notes: On the Macintosh, accelerators are normal in appearance, and the accelerator combination Command+*letter* is used.

Macintosh

Put (statement)

Syntax Put [#]filename, [recordnumber], variable

Description Writes data from the specified variable to a Random or Binary file.

Comments The `Put` statement accepts the following parameters:

Parameter	Description
<i>filenumber</i>	Integer representing the file to be written to. This is the same value as returned by the <code>Open</code> statement.
<i>recordnumber</i>	<p>Long specifying which record is to be written to the file.</p> <p>For <code>Binary</code> files, this number represents the first byte to be written starting with the beginning of the file (the first byte is 1). For <code>Random</code> files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647.</p> <p>If the <i>recordnumber</i> parameter is omitted, the next record is written to the file (if no records have been written yet, then the first record in the file is written). When <i>recordnumber</i> is omitted, the commas must still appear, as in the following example:</p> <pre>Put #1,,recvar</pre> <p>If <i>recordlength</i> is specified, it overrides any previous change in file position specified with the <code>Seek</code> statement.</p>

The *variable* parameter is the name of any variable of any of the following types:

Variable Type	File Storage Description
Integer	2 bytes are written to the file.
Long	4 bytes are written to the file.
String (variable-length)	<p>In <code>Binary</code> files, variable-length strings are written by first determining the specified string variable's length, then writing that many bytes to the file.</p> <p>In <code>Random</code> files, variable-length strings are written by first writing a 2-byte length, then writing that many characters to the file.</p>
String (fixed-length)	Fixed-length strings are written to <code>Random</code> and <code>Binary</code> files in the same way: the number of characters equal to the string's declared length are written.
Double	8 bytes are written to the file (IEEE format).
Single	4 bytes are written to the file (IEEE format).
Date	8 bytes are written to the file (IEEE double format).
Boolean	2 bytes are written to the file (either <code>-1</code> for <code>True</code> or <code>0</code> for <code>False</code>).
Variant	<p>A 2-byte <code>VarType</code> is written to the file followed by the data as described above. With variants of type <code>10</code> (user-defined errors), the 2-byte <code>VarType</code> is followed by a 2-byte unsigned integer (the error value), which is then followed by 2 additional bytes of information.</p> <p>The exception is with strings, which are always preceded by a 2-byte string length.</p>

User-defined types	<p>Each member of a user-defined data type is written individually.</p> <p>In Binary files, variable-length strings within user-defined types are written by first writing a 2-byte length followed by the string's content. This storage is different than variable-length strings outside of user-defined types.</p> <p>When writing user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.</p>
Arrays	Arrays cannot be written to a file using the Put statement.
Objects	Object variables cannot be written to a file using the Put statement.

With Random files, a runtime error will occur if the length of the data being written exceeds the record length (specified as the *reclen* parameter with the Open statement). If the length of the data being written is less than the record length, the entire record is written along with padding (whatever data happens to be in the I/O buffer at that time). With Binary files, the data elements are written contiguously: they are never separated with padding.

Example 'This example opens a file for random write, then writes ten records into the file with the values 10-50. Then the file is closed and reopened in random mode for read, and the records are read with the Get statement. The result is displayed in a dialog box.

```
Sub Main()  
  Open "test.dat" For Random Access Write As #1  
  For x = 1 To 10  
    r% = x * 10  
    Put #1,x,r%  
  Next x  
  Close  
  
  Open "test.dat" For Random Access Read As #1  
  For x = 1 To 10  
    Get #1,x,r%  
    msg = msg & "Record " & x & " is: " & r% & Basic.Eoln$  
  Next x  
  
  MsgBox msg  
  Close  
  Kill "test.dat"  
End Sub
```

See Also Open (statement); Put (statement); Write# (statement); Print# (statement).

Platform(s) Windows and Macintosh.

Pv (function)

Syntax `Pv(Rate,NPer,Pmt,Fv,Due)`

Description Calculates the present value of an annuity based on future periodic fixed payments and a constant rate of interest.

Comments The Pv function requires the following parameters:

Parameter	Description
<i>Rate</i>	Double representing the interest rate per period. When used with monthly payments, be sure to normalize annual percentage rates by dividing them by 12.
<i>NPer</i>	Double representing the total number of payments in the annuity.
<i>Pmt</i>	Double representing the amount of each payment per period.
<i>Fv</i>	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be 0.
<i>Due</i>	Integer indicating when the payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.

Rate and *NPer* must be expressed in the same units. If *Rate* is expressed in months, then *NPer* must also be expressed in months.

Positive numbers represent cash received, whereas negative numbers represent cash paid out.

Example 'This example demonstrates the present value (the amount you'd have to pay
'now) for a \$100,000 annuity that pays an annual income of \$5,000 over
20
'years at an annual interest rate of 10%.

```
Sub Main()
    pval = Pv(.1,20,-5000,100000,1)
    MsgBox "The present value is: " & Format(pval,"Currency")
End Sub
```

See Also Fv (function); IRR (function); MIRR (function); Npv (function).

Platform(s) Windows and Macintosh.

QueueEmpty (statement)

Syntax `QueueEmpty`

Description Empties the current event queue.

Comments After this statement, `QueFlush` will do nothing.

Example 'This code begins a new queue, then drags a selection over a range of characters in Notepad.

```
Sub Main()
    AppActivate "Notepad"
    QueEmpty 'Make sure the queue is empty.
    QueMouseDown ebLeftButton,1440,1393
    QueMouseUp ebLeftButton,4147,2363
    QueFlush True
End Sub
```

Platform(s) Windows.

QueFlush (statement)

Syntax `QueFlush isSaveState`

Description Plays back events that are stored in the current event queue.

Comments After `QueFlush` is finished, the queue is empty.

If *isSaveState* is `True`, then `QueFlush` saves the state of the Caps Lock, Num Lock, Scroll Lock, and Insert and restores the state after the `QueFlush` is complete. If this parameter is `False`, these states are not restored.

The function does not return until the entire queue has been played.

Example 'This example pumps some keys into Notepad.

```
Sub Main()
    AppActivate "Notepad"
    QueKeys "This is a test{Enter}"
    QueFlush True 'Play back the queue.
End Sub
```

Platform(s) Windows.

Platform The `QueFlush` statement uses the Windows journaling mechanism to replay the mouse and keyboard events stored in the queue. As a result, the mouse position may be changed. Furthermore, events can be played into any Windows application, including DOS applications running in a window.

Notes:

Windows

QueKeyDn (statement)

Syntax `QueKeyDn KeyString$ [,time]`

Description Appends key-down events for the specified keys to the end of the current event queue.

Comments The QueKeyDn statement accepts the following parameters:

Parameter	Description
-----------	-------------

<i>KeyString\$</i>	String containing the keys to be sent. The format for <i>KeyString\$</i> is described under the SendKeys statement.
--------------------	---

<i>time</i>	Integer specifying the number of milliseconds devoted for the output of the entire <i>KeyString\$</i> parameter. It must be within the following range:
-------------	---

$$0 \leq time \leq 32767$$

For example, if *time* is 5000 (5 seconds) and the *KeyString\$* parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.

The QueFlush command is used to play back the events stored in the current event queue.

Example 'This example plays back a Ctrl + mouse click.

```
Sub Main()  
  QueEmpty  
  QueKeyDn "^"  
  QueMouseClicked ebLeftButton 1024,792  
  QueKeyUp "^"  
  QueFlush True  
End Sub
```

See Also DoKeys (statement); SendKeys (statement); QueKeys (statement); QueKeyUp (statement); QueFlush (statement).

Platform(s) Windows.

QueKeys (statement)

Syntax QueKeys *KeyString\$* [, *time*]

Description Appends keystroke information to the current event queue.

Comments The QueKeys statement accepts the following parameters:

Parameter	Description
-----------	-------------

<i>KeyString\$</i>	String containing the keys to be sent. The format for <i>KeyString\$</i> is described under the SendKeys statement.
--------------------	---

time Integer specifying the number of milliseconds devoted for the output of the entire *KeyString\$* parameter. It must be within the following range:

$$0 \leq time \leq 32767$$

For example, if *time* is 5000 (5 seconds) and the *KeyString\$* parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.

The `QueFlush` command is used to play back the events stored in the current event queue.

Example

```
Sub Main()  
  WinActivate "Notepad"  
  QueEmpty  
  QueKeys "This is a test.{Enter}This is on a new line.{Enter}"  
  QueKeys "{Tab 3}This is indented with three tabs."  
  QueKeys "Some special characters: {~}{^}{%}{+}~"  
  QueKeys "Invoking the Find dialog.%Sf" 'Alt+S,F  
  QueFlush True  
End Sub
```

See Also `DoKeys (statement)`; `SendKeys (statement)`; `QueKeyDn (statement)`; `QueKeyUp (statement)`; `QueFlush (statement)`.

Platform(s) Windows.

Platform Notes: Under Windows, you cannot send keystrokes to MS-DOS applications running in a window.

Windows

QueKeyUp (statement)

Syntax `QueKeyUp KeyString$ [,time]`

Description Appends key-up events for the specified keys to the end of the current event queue.

Comments The `QueKeyUp` statement accepts the following parameters:

Parameter	Description
<i>KeyString\$</i>	String containing the keys to be sent. The format for <i>KeyString\$</i> is described under the <code>SendKeys</code> statement.

time Integer specifying the number of milliseconds devoted for the output of the entire *KeyString\$* parameter. It must be within the following range:

$$0 \leq time \leq 32767$$

For example, if *time* is 5000 (5 seconds) and the *KeyString\$* parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.

The `QueFlush` command is used to play back the events stored in the current event queue.

Example See `QueKeyDn` (statement).

See Also `DoKeys` (statement); `SendKeys` (statement); `QueKeys` (statement); `QueKeyDn` (statement); `QueFlush` (statement).

Platform(s) Windows.

QueMouseClicked (statement)

Syntax `QueMouseClicked button,X,Y [,time]`

Description Adds a mouse click to the current event queue.

Comments The `QueMouseClicked` statement takes the following parameters:

Parameter	Description
-----------	-------------

<i>button</i>	Integer specifying which mouse button to click:
---------------	---

`ebLeftButton` Click the left mouse button.

`ebRightButton` Click the right mouse button.

<i>X, Y</i>	Integer coordinates, in twips, where the mouse click is to be recorded.
-------------	---

<i>time</i>	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse click will play back at full speed.
-------------	---

A mouse click consists of a mouse button down at position *X, Y*, immediately followed by a mouse button up.

The `QueFlush` command is used to play back the events stored in the current event queue.

Example 'This example activates Notepad and invokes the Find dialog box. It then
'uses the QueMouseClicked command to click the Cancel button.

```
Sub Main()  
    AppActivate "Notepad" 'Activate Notepad.  
    QueKeys "%Sf" 'Invoke the Find dialog box.  
    QueFlush True 'Play this back (allow dialog box to open).  
    QueSetRelativeWindow 'Set mouse relative to Find dialog box.  
    QueMouseClicked ebLeftButton,7059,1486 'Click the Cancel button.  
    QueFlush True 'Play back the queue.  
End Sub
```

See Also QueMouseDown (statement); QueMouseUp (statement); QueMouseDownClick (statement); QueMouseDownClick (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

Platform(s) Windows.

QueMouseDownClick (statement)

Syntax QueMouseDownClick *button*, *X*, *Y* [, *time*]

Description Adds a mouse double click to the current event queue.

Comments The QueMouseDownClick statement takes the following parameters:

Parameter	Description
<i>button</i>	Integer specifying which mouse button to double-click: <div style="margin-left: 40px;">ebLeftButton Double-click the left mouse button. ebRightButton Double-click the right mouse button.</div>
<i>X</i> , <i>Y</i>	Integer coordinates, in twips, where the mouse double click is to be recorded.
<i>time</i>	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse double click will play back at full speed. A mouse double click consists of a mouse down/up/down/up at position <i>X</i> , <i>Y</i> . The events are queued in such a way that a double click is registered during queue playback. The QueFlush command is used to play back the events stored in the current event queue.

Example 'This example double-clicks the left mouse button.
QueMouseDownClick ebLeftButton,344,360

See Also QueMouseClicked (statement); QueMouseDown (statement); QueMouseUp (statement); QueMouseDown (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

Platform(s) Windows.

QueMouseDown (statement)

Syntax QueMouseDown *button*, *X*, *Y* [, *time*]

Description Adds a mouse double down to the end of the current event queue.

Comments The QueMouseDown statement takes the following parameters:

Parameter	Description
<i>button</i>	Integer specifying which mouse button to press: <div style="margin-left: 40px;">ebLeftButton Click the left mouse button.</div> <div style="margin-left: 40px;">ebRightButton Click the right mouse button.</div>
<i>x,y</i>	Integer coordinates, in twips, where the mouse double down is to be recorded.
<i>time</i>	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse double down will play back at full speed. This statement adds a mouse double down to the current event queue. A double down consists of a mouse down/up/down at position <i>X</i> , <i>Y</i> . The QueFlush command is used to play back the events stored in the current event queue.

Example 'This example double-clicks a word, then drags it to a new location.

```
Sub Main()
    QueFlush                                'Start with empty queue.
    QueMouseDown ebLeftButton,356,4931     'Double-click, mouse still
down.
    QueMouseMove 600,4931                    'Drag to new spot.
    QueMouseUp ebLeftButton                 'Now release the mouse.
    QueFlush True                           'Play back the queue.
End Sub
```

See Also QueMouseClicked (statement); QueMouseDown (statement); QueMouseUp (statement); QueMouseDown (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

Platform(s) Windows.

QueMouseDown (statement)

Syntax `QueMouseDown button,X,Y [,time]`

Description Adds a mouse down to the current event queue.

Comments The `QueMouseDown` statement takes the following parameters:

Parameter	Description
<i>button</i>	Integer specifying which mouse button to press: <div style="margin-left: 40px;"><code>ebLeftButton</code> Click the left mouse button.</div> <div style="margin-left: 40px;"><code>ebRightButton</code> Click the right mouse button.</div>
<i>X, Y</i>	Integer coordinates, in twips, where the mouse down is to be recorded.
<i>time</i>	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse down will play back at full speed. The <code>QueFlush</code> command is used to play back the events stored in the current event queue.

Example See `QueEmpty` (statement).

See Also `QueMouseClicked` (statement); `QueMouseUp` (statement); `QueMouseDownClick` (statement); `QueMouseDownDn` (statement); `QueMouseMove` (statement); `QueMouseMoveBatch` (statement); `QueFlush` (statement).

Platform(s) Windows.

QueMouseMove (statement)

Syntax `QueMouseMove X,Y [,time]`

Description Adds a mouse move to the current event queue.

Comments The `QueMouseMove` statement takes the following parameters:

Parameter	Description
<i>X, Y</i>	Integer coordinates, in twips, where the mouse is to be moved.
<i>time</i>	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse move will play back at full speed. The <code>QueFlush</code> command is used to play back the events stored in the current event queue.

Example See QueMouseDown (statement).

See Also QueMouseClicked (statement); QueMouseDown (statement); QueMouseUp (statement); QueMouseDownClick (statement); QueMouseDown (statement); QueMouseMoveBatch (statement); QueFlush (statement).

Platform(s) Windows.

QueMouseMoveBatch (statement)

Syntax QueMouseMoveBatch *ManyMoves\$*

Description Adds a series of mouse-move events to the current event queue.

Comments The *ManyMoves\$* parameter is a string containing positional and timing information in the following format:

X,Y,time [,X,Y,time] . . .

The *X* and *Y* parameters specify a mouse position in twips. The *time* parameter specifies the delay in milliseconds between the current mouse move and the previous event in the queue. If *time* is 0, then the mouse move will play back as fast as possible.

The QueMouseMoveBatch command should be used in place of a series of QueMouseMove statements to reduce the number of lines in your script. A further advantage is that, since the mouse-move information is contained within a literal string, the storage for the data is placed in the constant segment instead of the code segment, reducing the size of the code.

The QueFlush command is used to play back the events stored in the current event queue.

Example 'This example activates PaintBrush, then paints the word "Hi".

```
Sub Main()  
    AppActivate "Paintbrush"  
    AppMaximize  
    QueMouseDown ebLeftButton,2175,3412  
    QueMouseMoveBatch  
    "2488,3224,0,2833,2786,0,3114,2347,0,3208,2160,0,3240,2097,0"  
    QueMouseMoveBatch  
    "3255,2034,0,3255,1987,0,3255,1956,0,3255,1940,0,3224,1956,0"  
    QueMouseMoveBatch  
    "3193,1987,0,3114,2019,0,3036,2066,0,3005,2113,0,2973,2175,0"  
    QueMouseMoveBatch  
    "2942,2332,0,2926,2394,0,2926,2582,0,2911,2739,0,2911,2801,0"  
    QueMouseMoveBatch  
    "2911,2958,0,2911,3020,0,2911,3052,0,2911,3083,0,2911,3114,0"  
    QueMouseMoveBatch  
    "2911,3130,0,2895,3161,0,2895,3193,0,2895,3208,0,2895,3193,0"  
    QueMouseMoveBatch  
    "2895,3146,0,2911,3083,0,2926,3020,0,2942,2958,0,2973,2895,0"  
    QueMouseMoveBatch  
    "3005,2848,0,3020,2817,0,3036,2801,0,3052,2770,0,3083,2770,0"  
    QueMouseMoveBatch  
    "3114,2754,0,3130,2754,0,3146,2770,0,3161,2786,0,3161,2848,0"  
    QueMouseMoveBatch  
    "3193,3005,0,3193,3193,0,3208,3255,0,3224,3318,0,3240,3349,0"  
    QueMouseMoveBatch  
    "3255,3349,0,3286,3318,0,3380,3271,0,3474,3208,0,3553,3052,0"  
    QueMouseMoveBatch  
    "3584,2895,0,3615,2739,0,3631,2692,0,3631,2645,0,3646,2645,0"  
    QueMouseMoveBatch  
    "3646,2660,0,3646,2723,0,3646,2880,0,3662,2942,0,3693,2989,0"  
    QueMouseMoveBatch "3709,3005,0,3725,3005,0,3756,2989,0,3787,2973,0"  
    QueMouseUp ebLeftButton,3787,2973  
    QueMouseDown ebLeftButton,3678,2535  
    QueMouseMove 3678,2520  
    QueMouseMove 3678,2535  
    QueMouseUp ebLeftButton,3678,2535  
    QueFlush True  
End Sub
```

See Also QueMouseClicked (statement); QueMouseDown (statement); QueMouseUp (statement); QueMouseDblClk (statement); QueMouseDblDn (statement); QueMouseMove (statement); QueFlush (statement).

Platform(s) Windows.

QueMouseUp (statement)

Syntax QueMouseUp *button,X,Y [,time]*

Description Adds a mouse up to the current event queue.

Comments The QueMouseUp statement takes the following parameters:

Parameter	Description
-----------	-------------

<i>button</i>	Integer specifying the mouse button to be released: <table> <tr> <td>ebLeftButton</td><td>Release the left mouse button.</td></tr> <tr> <td>ebRightButton</td><td>Release the right mouse button.</td></tr> </table>	ebLeftButton	Release the left mouse button.	ebRightButton	Release the right mouse button.
ebLeftButton	Release the left mouse button.				
ebRightButton	Release the right mouse button.				
<i>X, Y</i>	Integer coordinates, in twips, where the mouse button is to be released.				
<i>time</i>	Integer specifying the delay in milliseconds between this event and the previous event in the queue. If this parameter is omitted (or 0), the mouse up will play back at full speed. The QueFlush command is used to play back the events stored in the current event queue.				

Example See QueEmpty (statement).

See Also QueMouseClicked (statement); QueMouseDown (statement); QueMouseDblClk (statement); QueMouseDblDn (statement); QueMouseMove (statement); QueMouseMoveBatch (statement); QueFlush (statement).

Platform(s) Windows.

QueSetRelativeWindow (statement)

Syntax QueSetRelativeWindow [*window_object*]

Description Forces all subsequent QueX commands to adjust the mouse positions relative to the specified window.

Comments The *window_object* parameter is an object of type HWND. If *window_object* is Nothing or omitted, then the window with the focus is used (i.e., the active window).

The QueFlush command is used to play back the events stored in the current event queue.

Example

```
Sub Main()
    'Adjust mouse coordinates relative to Notepad.
    Dim a As HWND
    Set a = WinFind("Notepad")
    QueSetRelativeWindow a
End Sub
```

Platform(s) Windows.

Random (function)

Syntax Random(*min*, *max*)

Description Returns a Long value greater than or equal to *min* and less than or equal to *max*.

Comments Both the *min* and *max* parameters are rounded to Long. A runtime error is generated if *min* is greater than *max*.

Example 'This example uses the random number generator to generate ten 'lottery numbers.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Randomize                                'Start with new random seed.
    For x = 1 To 10
        y = Random(0,100)                    'Generate numbers.
        msg = msg & y & crlf
    Next x
    MsgBox "Ten numbers for the lottery: " & crlf & msg
End Sub
```

See Also Randomize (statement); Random (function).

Platform(s) Windows and Macintosh.

Randomize (statement)

Syntax Randomize [*seed*]

Description Initializes the random number generator with a new seed.

Comments If *seed* is not specified, then the current value of the system clock is used.

Example 'This example sets the randomize seed to a random number between '100 and 1000, then generates ten random numbers for the lottery.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Randomize                                'Start with new random seed.
    For x = 1 To 10
        y = Random(0,100)                    'Generate numbers.
        msg = msg + Str(y) + crlf
    Next x
    MsgBox "Ten numbers for the lottery: " & crlf & msg
End Sub
```

See Also Random (function); Rnd (function).

Platform(s) Windows and Macintosh.

Rate (function)

Syntax `Rate(NPer,Pmt,Pv,Fv,Due,Guess)`

Description Returns the rate of interest for each period of an annuity.

Comments An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans.

The `Rate` function requires the following parameters:

Parameter	Description
<i>NPer</i>	Double representing the total number of payments in the annuity.
<i>Pmt</i>	Double representing the amount of each payment per period.
<i>Pv</i>	Double representing the present value of your annuity. In a loan situation, the present value would be the amount of the loan.
<i>Fv</i>	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be zero.
<i>Due</i>	Integer specifying when the payments are due for each payment period. A 0 indicates payment at the end of each period, whereas a 1 indicates payment at the start of each period.
<i>Guess</i>	<p>Double specifying a guess as to the value the <code>Rate</code> function will return. The most common guess is .1 (10 percent).</p> <p>Positive numbers represent cash received, whereas negative values represent cash paid out.</p> <p>The value of <i>Rate</i> is found by iteration. It starts with the value of <i>Guess</i> and cycles through the calculation adjusting <i>Guess</i> until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, <i>Rate</i> fails, and the user must pick a better guess.</p>

Example 'This example calculates the rate of interest necessary to save \$8,000
'by paying \$200 each year for 48 years. The guess rate is 10%.

```
Sub Main()
    r# = Rate(48,-200,8000,0,1,.1)
    MsgBox "The rate required is: " & Format(r#,"Percent")
End Sub
```

See Also `IPmt (function)`; `NPer (function)`; `Pmt (function)`; `PPmt (function)`.

Platform(s) Windows and Macintosh.

ReadIni\$ (function)

Syntax `ReadIni$(section$, item$[, filename$])`

Description Returns a String containing the specified item from an ini file.

Comments The ReadIni\$ function takes the following parameters:

Parameter	Description
<i>section\$</i>	String specifying the section that contains the desired variable, such as "windows". Section names are specified without the enclosing brackets.
<i>item\$</i>	String specifying the item whose value is to be retrieved.
<i>filename\$</i>	String containing the name of the ini file to read.

See Also WriteIni (statement); ReadIniSection (statement).

Platform(s) Windows.

Platform Notes: Under Windows, if the name of the ini file is not specified, then win.ini is assumed.

Windows If the *filename\$* parameter does not include a path, then this statement looks for ini files in the Windows directory.

ReadIniSection (statement)

Syntax `ReadIniSection section$,ArrayOfItems()[, filename$]`

Description Fills an array with the item names from a given section of the specified ini file.

Comments The ReadIniSection statement takes the following parameters:

Parameter	Description
-----------	-------------

<i>section\$</i>	String specifying the section that contains the desired variables, such as "windows". Section names are specified without the enclosing brackets.
------------------	---

<i>ArrayOfItems()</i>	Specifies either a zero- or a one-dimensioned array of strings or variants. The array can be either dynamic or fixed.
-----------------------	---

If *ArrayOfItems()* is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound, UBound, and ArrayDims functions to determine the number and size of the new array's dimensions.

If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.

<i>filename\$</i>	String containing the name of an ini file.
-------------------	--

On return, the *ArrayOfItems()* parameter will contain one array element for each variable in the specified ini section.

Example

```
Sub Main()
    Dim items() As String
    ReadIniSection "windows", items$
    r% = SelectBox("INI Items", , items$)
End Sub
```

See Also ReadIni\$ (function); WriteIni (statement).

Platform(s) Windows.

Platform Notes: Under Windows, if the name of the ini file is not specified, then win.ini is assumed.

Windows If the *filename\$* parameter does not include a path, then this statement looks for ini files in the Windows directory.

Redim (statement)

Syntax Redim [Preserve] *variablename* (*subscriptRange*) [As *type*], ...

Description Redimensions an array, specifying a new upper and lower bound for each dimension of the array.

Comments The *variablename* parameter specifies the name of an existing array (previously declared using the Dim statement) or the name of a new array variable. If the array variable already exists, then it must previously have been declared with the Dim statement with no dimensions, as shown in the following example:

```
Dim a$() 'Dynamic array of strings (no dimensions yet)
```

Dynamic arrays can be redimensioned any number of times.

The *subscriptRange* parameter specifies the new upper and lower bounds for each dimension of the array using the following syntax:

```
[lower To] upper [, [lower To] upper]...
```

If *lower* is not specified, then 0 is used (or the value set using the Option Base statement). A runtime error is generated if *lower* is less than *upper*. Array dimensions must be within the following range:

```
-32768 <= lower <= upper <= 32767
```

The *type* parameter can be used to specify the array element type. Arrays can be declared using any fundamental data type, user-defined data types, and objects.

Redimensioning an array erases all elements of that array unless the Preserve keyword is specified. When this keyword is specified, existing data in the array is preserved where possible. If the number of elements in an array dimension is increased, the new elements are initialized to 0 (or empty string). If the number of elements in an array dimension is decreased, then the extra elements will be deleted. If the Preserve keyword is specified, then the number of dimensions of the array being redimensioned must either be zero or the same as the new number of dimensions.

Example 'This example uses the FileList statement to redim an array and fill it 'with filename strings. A new array is then redimmed to hold the 'number of elements found by FileList, and the FileList array is 'copied into it and partially displayed.

```
Sub Main()  
    Dim fl$()  
    FileList fl$, "*. *"   
    count = Ubound(fl$)  
    Redim nl$(Lbound(fl$) To Ubound(fl$))  
    For x = 1 to count  
        nl$(x) = fl(x)  
    Next x  
    MsgBox "The last element of the new array is: " & nl$(count)  
End Sub
```

See Also Dim (statement); Public (statement); Private (statement); ArrayDims (function); LBound (function); UBound (function).

Platform(s) Windows and Macintosh.

Rem (statement)

Syntax `Rem text`

Description Causes the compiler to skip all characters on that line.

Example

```
Sub Main()
    Rem This is a line of comments that serves to illustrate the
    Rem workings of the code. You can insert comments to make it more
    Rem readable and maintainable in the future.
End Sub
```

See Also ' (keyword); Comments (topic).

Platform(s) Windows and Macintosh.

Reset (statement)

Syntax `Reset`

Description Closes all open files, writing out all I/O buffers.

Example

```
'This example opens a file for output, closes it with the Reset
statement,
'then deletes it with the Kill statement.

Sub Main()
    Open "test.dat" for Output Access Write as # 1
    Reset
    Kill "test.dat"

    If FileExists("test.dat") Then
        MsgBox "The file was not deleted."
    Else
        MsgBox "The file was deleted."
    End If
End Sub
```

See Also Close (statement); Open (statement).

Platform(s) Windows and Macintosh.

Resume (statement)

Syntax `Resume {[0] | Next | label}`

Description Ends an error handler and continues execution.

Comments The form `Resume 0` (or simply `Resume` by itself) causes execution to continue with the statement that caused the error.

The form `Resume Next` causes execution to continue with the statement following the statement that caused the error.

The form `Resume label` causes execution to continue at the specified label.

The `Resume` statement resets the error state. This means that, after executing this statement, new errors can be generated and trapped as normal.

Example 'This example accepts two integers from the user and attempts to multiply the numbers together. If either number is larger than an integer, the program processes an error routine and then continues program execution at a specific section using `Resume <label>`'. Another error trap is then set using `Resume Next`. The new error trap will clear any previous error branching and also 'tell' the program to continue execution of the program even if an error is encountered.

```
Sub Main()  
    Dim a%, b%, x%  
  
    Again:  
        On Error Goto Overflow  
        a% = InputBox("Enter 1st integer to multiply", "Enter Number")  
        b% = InputBox("Enter 2nd integer to multiply", "Enter Number")  
  
        On Error Resume Next          'Continue program execution at next line    x% = a%  
  
        if err = 0 then  
            MsgBox x%  
        else  
            MsgBox a% & " * " & b% & " cause an overflow!"  
        end if  
  
        Exit Sub  
  
Overflow:                          'Error handler.  
    MsgBox "You've entered a non-integer value, try again!"  
    Resume Again  
End Sub
```

See Also Error Handling (topic); On Error (statement).

Platform(s) Windows and Macintosh.

Return (statement)

Syntax `Return`

Description Transfers execution control to the statement following the most recent `GoSub`.

Comments A runtime error results if a Return statement is encountered without a corresponding GoSub statement.

Example 'This example calls a subroutine and then returns execution to the Main routine by the Return statement.

```
Sub Main()
    GoSub SubTrue
    MsgBox "The Main routine continues here."
    Exit Sub

SubTrue:
    MsgBox "This message is generated in the subroutine."
    Return
    Exit Sub
End Sub
```

See Also GoSub (statement).

Platform(s) Windows and Macintosh.

Right, Right\$ (functions)

Syntax Right[\$](*text*, *NumChars*)

Description Returns the rightmost *NumChars* characters from a specified string.

Comments Right\$ returns a String, whereas Right returns a String variant. The Right function takes the following parameters:

Parameter	Description
<i>text</i>	String from which characters are returned. A runtime error is generated if <i>text</i> is Null.
<i>NumChars</i>	Integer specifying the number of characters to return. If <i>NumChars</i> is greater than or equal to the length of the string, then the entire string is returned. If <i>NumChars</i> is 0, then a zero-length string is returned.

Example 'This example shows the Right\$ function used in a routine to change uppercase names to lowercase with an uppercase first letter.

```
Sub Main()
    lname$ = "WILLIAMS"
    x = Len(lname$)
    rest$ = Right$(lname$, x - 1)
    fl$ = Left$(lname$, 1)
    lname$ = fl$ & LCase$(rest$)
    MsgBox "The converted name is: " & lname$
End Sub
```

See Also Left, Left\$ (functions).

Platform(s) Windows and Macintosh.

RmDir (statement)

Syntax `RmDir dir$`

Comments Removes the directory specified by the `String` contained in *dir*\$.

Example 'This routine creates a directory and then deletes it with `RmDir`.

```
Sub Main()  
    On Error Goto ErrMake  
    MkDir("test01")  
    On Error Goto ErrRemove  
    RmDir("test01")  
  
ErrMake:  
    MsgBox "The directory could not be created."  
    Exit Sub  
  
ErrRemove:  
    MsgBox "The directory could not be removed."  
    Exit Sub  
End Sub
```

See Also `ChDir (statement)`; `ChDrive (statement)`; `CurDir, CurDir$ (functions)`; `Dir, Dir$ (functions)`; `MkDir (statement)`.

Platform(s) Windows and Macintosh.

Platform Under Windows, this command behaves the same as the DOS "rd" command.

Notes:
Windows

Rnd (function)

Syntax `Rnd[(number)]`

Description Returns a random `Single` number between 0 and 1.

Comments If *number* is omitted, the next random number is returned. Otherwise, the *number* parameter has the following meaning:

If	Then
<i>number</i> < 0	Always returns the same number.
<i>number</i> = 0	Returns the last number generated.
<i>number</i> > 0	Returns the next random number.

Example 'This routine generates a list of random numbers and displays them.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    For x = -1 To 8
        y! = Rnd(1) * 100
        msg = msg & x & " : " & y! & crlf
    Next x
    MsgBox msg & "Last form: " & Rnd
End Sub
```

See Also Randomize (statement); Random (function).

Platform(s) Windows and Macintosh.

RSet (statement)

Syntax RSet *destvariable* = *source*

Description Copies the source string *source* into the destination string *destvariable*.

Comments If *source* is shorter in length than *destvariable*, then the string is right-aligned within *destvariable* and the remaining characters are padded with spaces. If *source* is longer in length than *destvariable*, then *source* is truncated, copying only the leftmost number of characters that will fit in *destvariable*. A runtime error is generated if *source* is Null.

The *destvariable* parameter specifies a String or Variant variable. If *destvariable* is a Variant containing Empty, then no characters are copied. If *destvariable* is not convertible to a String, then a runtime error occurs. A runtime error results if *destvariable* is Null.

Example 'This example replaces a 40-character string of asterisks (*) with 'an RSet and LSet string and then displays the result.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    Dim msg,tmpstr$
    tmpstr$ = String$(40, "*")
    msg = "Here are two strings that have been right-" & crlf
    msg = msg & "and left-justified in a 40-character string."
    msg = msg & crlf & crlf
    RSet tmpstr$ = "Right->"
    msg = msg & tmpstr$ & crlf
    LSet tmpstr$ = "<-Left"
    msg = msg & tmpstr$ & crlf
    MsgBox msg
End Sub
```

See Also LSet (statement).

Platform(s) Windows and Macintosh.

RTrim, RTrim\$ (functions)

Syntax RTrim[\$](*text*)

Description Returns a string with the trailing spaces removed.

Comments RTrim\$ returns a String, whereas RTrim returns a String variant.

Null is returned if *text* is Null.

Example 'This example displays a left-justified string and its RTrim result.

```
Const crlf = Chr$(13) + Chr$(10)
```

```
Sub Main()  
    a$ = "This is a left-justified string.           "  
    b$ = RTrim$(a$)  
    MsgBox a$ & "<=" & crlf & b$ & "<=" & "  
End Sub
```

See Also LTrim, LTrim\$ (functions); Trim, Trim\$ (functions).

Platform(s) Windows and Macintosh.

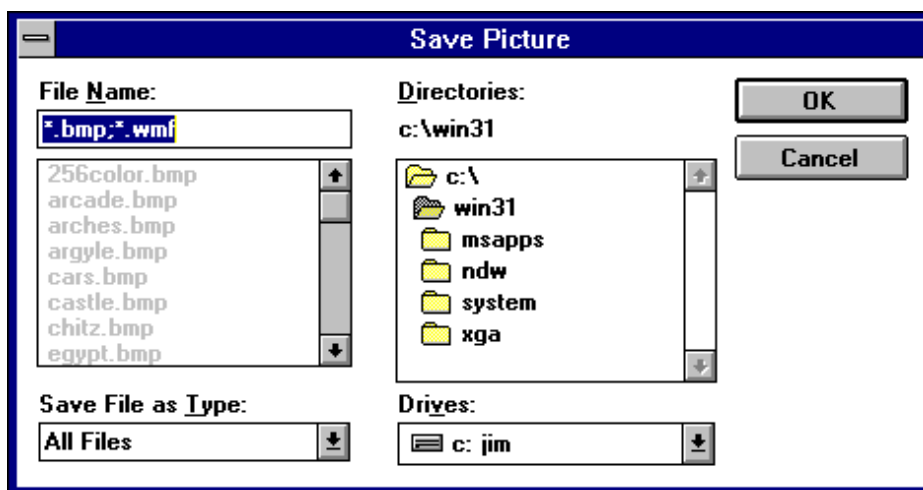
SaveFilename\$ (function)

Syntax SaveFilename\$([title\$ [,extensions\$]])

Description Displays a dialog box that prompts the user to select from a list of files and returns a String containing the full path of the selected file.

Comments The SaveFilename\$ function accepts the following parameters:

Parameter	Description
<i>title\$</i>	String containing the title that appears on the dialog box's caption. If this string is omitted, then "Save As" is used.
<i>extensions\$</i>	String containing the available file types. Its format depends on the platform on which WM Basic is running. If this string is omitted, then all files are used. The SaveFilename\$ function returns a full pathname of the file that the user selects. A zero-length string is returned if the user selects Cancel. If the file already exists, then the user is prompted to overwrite it. e\$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF" f\$ = SaveFilename\$ ("Save Picture",e\$)



Example 'This example creates a save dialog box, giving the user the ability to save to several different file types.

```
Sub Main()
    e$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF"
    f$ = SaveFilename$("Save Picture",e$)
    If Not f$ = "" Then
        MsgBox "User choose to save file as: " + f$
    Else
        MsgBox "User canceled."
    End IF
End Sub
```

See Also **MsgBox** (statement); **AskBox\$** (function); **AskPassword\$** (function); **InputBox**, **InputBox\$** (functions); **OpenFilename\$** (function); **SelectBox** (function); **AnswerBox** (function).

Platform(s) Windows and Macintosh.

Platform Notes: Under Windows, the *extensions\$* parameter must be in the following format:

description:ext[,ext][;description:ext[,ext]]...

Windows Placeholder

Description

description Specifies the grouping of files for the user, such as All Files.

ext Specifies a valid file extension, such as *.BAT or *.*F?.

For example, the following are valid *extensions\$* specifications:

```
"All Files:*"
"Documents:*.TXT,*.DOC"
"All Files:*;Documents:*.TXT,*.DOC"
```

Platform Notes: On the Macintosh, the *extensions\$* parameter contains a comma-separated list of four-character file types. For example:

Macintosh "TEXT,XLS4,MSWD"

On the Macintosh, the *title\$* parameter is ignored.

Screen.DlgBaseUnitsX (property)

Syntax Screen.DlgBaseUnitsX

Description Returns an Integer used to convert horizontal pixels to and from dialog units.

Comments The number returned depends on the name and size of the font used to display dialog boxes.

To convert from pixels to dialog units in the horizontal direction:

$((X\text{Pixels} * 4) + (\text{Screen.DlgBaseUnitsX} - 1)) / \text{Screen.DlgBaseUnitsX}$

To convert from dialog units to pixels in the horizontal direction:

$(X\text{DlgUnits} * \text{Screen.DlgBaseUnitsX}) / 4$

Example 'This example converts the screen width from pixels to dialog units.

```
Sub Main()  
    XPixels = Screen.Width  
    conv% = Screen.DlgBaseUnitsX  
    XDlgUnits = (XPixels * 4) + (conv% - 1) / conv%  
    MsgBox "The screen width is " & XDlgUnits & " dialog units."  
End Sub
```

See Also Screen.DlgBaseUnitsY (property).

Platform(s) Windows.

Screen.DlgBaseUnitsY (property)

Syntax Screen.DlgBaseUnitsY

Description Returns an Integer used to convert vertical pixels to and from dialog units.

Comments The number returned depends on the name and size of the font used to display dialog boxes.

To convert from pixels to dialog units in the vertical direction:

$(Y\text{Pixels} * 8) + (\text{Screen.DlgBaseUnitsY} - 1) / \text{Screen.DlgBaseUnitsY}$

To convert from dialog units to pixels in the vertical direction:

$(Y\text{DlgUnits} * \text{Screen.DlgBaseUnitsY}) / 8$

Example 'This example converts the screen width from pixels to dialog units.

```
Sub Main()  
    YPixels = Screen.Height  
    conv% = Screen.DlgBaseUnitsY  
    YDlgUnits = (YPixels * 8) + (conv% - 1) / conv%  
    MsgBox "The screen width is " & YDlgUnits & " dialog units."  
End Sub
```

See Also Screen.DlgBaseUnitsX (property).

Platform(s) Windows.

Screen.Height (property)

Syntax Screen.Height

- Description** Returns the height of the screen in pixels as an Integer.
- Comments** This property is used to retrieve the height of the screen in pixels. This value will differ depending on the display resolution.
- This property is read-only.
- Example** 'This example displays the screen height in pixels.
- ```
Sub Main()
 MsgBox "The Screen height is " & Screen.Height & " pixels."
End Sub
```
- See Also** Screen.Width (property).
- Platform(s)** Windows.

---

## Screen.TwipsPerPixelX (property)

---

- Syntax** Screen.TwipsPerPixelX
- Description** Returns an Integer representing the number of twips per pixel in the horizontal direction of the installed display driver.
- Comments** This property is read-only.
- Example** 'This example displays the number of twips across the screen horizontally.
- ```
Sub Main()  
    XScreenTwips = Screen.Width * Screen.TwipsPerPixelX  
    MsgBox "Total horizontal screen twips = " & XScreenTwips  
End Sub
```
- See Also** Screen.TwipsPerPixelY (property).
- Platform(s)** Windows.

Screen.TwipsPerPixelY (property)

- Syntax** Screen.TwipsPerPixelY
- Description** Returns an Integer representing the number of twips per pixel in the vertical direction of the installed display driver.
- Comments** This property is read-only.
- Example** 'This example displays the number of twips across the screen vertically.
- ```
Sub Main()
 YScreenTwips = Screen.Height * Screen.TwipsPerPixelY
 MsgBox "Total vertical screen twips = " & YScreenTwips
End Sub
```

**See Also** Screen.TwipsPerPixelX (property).

**Platform(s)** Windows.

## Screen.Width (property)

**Syntax** Screen.Width

**Description** Returns the width of the screen in pixels as an Integer.

**Comments** This property is used to retrieve the width of the screen in pixels. This value will differ depending on the display resolution.

This property is read-only.

**Example**

```
'This example displays the screen width in pixels
Sub Main()
 MsgBox "The screen width is " & Screen.Width & " pixels."
End Sub
```

**See Also** Screen.Height (property).

**Platform(s)** Windows.

## Second (function)

**Syntax** Second(*time*)

**Description** Returns the second of the day encoded in the specified *time* parameter.

**Comments** The value returned is an Integer between 0 and 59 inclusive.

The *time* parameter is any expression that converts to a Date.

**Example**

```
'This example takes the current time; extracts the hour,
'minute, and second; and displays them as the current time.
Sub Main()
 xt# = TimeValue(Time$())
 xh# = Hour(xt#)
 xm# = Minute(xt#)
 xs# = Second(xt#)
 MsgBox "The current time is: " & CStr(xh#) & ":" & CStr(xm#) & ":"
 & CStr(xs#)
End Sub
```

**See Also** Day (function); Minute (function); Month (function); Year (function); Hour (function); Weekday (function); DatePart (function).

**Platform(s)** Windows and Macintosh.

## Seek (function)

**Syntax** `Seek (filenumber)`

**Description** Returns the position of the file pointer in a file relative to the beginning of the file.

**Comments** The *filenumber* parameter is a number that WM Basic uses to refer to the open file—the number passed to the Open statement.

The value returned depends on the mode in which the file was opened:

| File Mode | Returns                                         |
|-----------|-------------------------------------------------|
| Input     | Byte position for the next read                 |
| Output    | Byte position for the next write                |
| Append    | Byte position for the next write                |
| Random    | Number of the next record to be written or read |
| Binary    | Byte position for the next read or write        |

The value returned is a Long between 1 and 2147483647, where the first byte (or first record) in the file is 1.

**Example** 'This example opens a file for random write, then writes ten 'records into the file using the PUT statement. The file position is 'displayed using the Seek Function, and the file is closed.

```
Sub Main()
 Open "test.dat" For Random Access Write As #1
 For x = 1 To 10
 r% = x * 10
 Put #1,x,r%
 Next x
 y = Seek(1)
 MsgBox "The current file position is: " & y
 Close
End Sub
```

**See Also** `Seek (statement)`; `Loc (function)`.

**Platform(s)** Windows and Macintosh.

## Seek (statement)

**Syntax** `Seek [#] filenumber, position`

**Description** Sets the position of the file pointer within a given file such that the next read or write operation will occur at the specified position.

**Comments** The Seek statement accepts the following parameters:

| Parameter         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenumber</i> | Integer used by WM Basic to refer to the open file—the number passed to the Open statement.                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>position</i>   | <p>Long that specifies the location within the file at which to position the file pointer. The value must be between 1 and 2147483647, where the first byte (or record number) in the file is 1. For files opened in either Binary, Output, Input, or Append mode, <i>position</i> is the byte position within the file. For Random files, <i>position</i> is the record number.</p> <p>A file can be extended by seeking beyond the end of the file and writing data there.</p> |

**Example** 'This example opens a file for random write, then writes ten 'records into the file using the PUT statement. The file is 'then reopened for read, and the ninth record is read using 'the Seek and Get functions.

```
Sub Main()
 Open "test.dat" For Random Access Write As #1
 For x = 1 To 10
 rec$ = "Record#: " & x
 Put #1,x,rec$
 Next x
 Close

 Open "test.dat" For Random Access Read As #1
 Seek #1,9
 Get #1,,rec$
 MsgBox "The ninth record = " & x
 Close
 Kill "test.dat"
End Sub
```

**See Also** Seek (function); Loc (function).

**Platform(s)** Windows and Macintosh.

## Select...Case (statement)

**Syntax** Select Case *testexpression*  
 [Case *expressionlist*  
     [*statement\_block*]]  
 [Case *expressionlist*  
     [*statement\_block*]]  
     .  
     .  
 [Case Else  
     [*statement\_block*]]  
 End Select

**Description** Used to execute a block of WM Basic statements depending on the value of a given expression.

**Comments** The `Select Case` statement has the following parts:

| Part                   | Description                                                                                                                                                                    |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>testexpression</i>  | Any numeric or string expression.                                                                                                                                              |
| <i>statement_block</i> | Any group of WM Basic statements. If the <i>testexpression</i> matches any of the expressions contained in <i>expressionlist</i> , then this statement block will be executed. |
| <i>expressionlist</i>  | A comma separated list of expressions to be compared against <i>testexpression</i> using any of the following syntaxes:                                                        |

*expression* [ , *expression* ] . . .  
*expression* `to` *expression*  
*is* *relational\_operator* *expression*

The resultant type of *expression* in *expressionlist* must be the same as that of *testexpression*.

Multiple expression ranges can be used within a single `Case` clause. For example:

```
Case 1 to 10,12,15, Is > 40
```

Only the *statement\_block* associated with the first matching expression will be executed. If no matching *statement\_block* is found, then the statements following the `Case Else` will be executed.

A `Select . . . End Select` expression can also be represented with the `If . . . Then` expression. The use of the `Select` statement, however, may be more readable.

**Example** 'This example uses the `Select . . . Case` statement to output the 'current operating system.

```
Sub Main()
 OpSystem% = Basic.OS
 Select Case OpSystem%
 Case 0
 s = "Microsoft Windows"
 Case 10
 s = "Macintosh"
 Case Else
 s = "Other"
 End Select
 MsgBox "This version of WM Basic is running on: " & s
End Sub
```



**See Also** Choose (function); Switch (function); IIf (function); If...Then...Else (statement).

**Platform(s)** Windows and Macintosh.

## SelectBox (function)

**Syntax** `SelectBox(title, prompt, ArrayOfItems)`

**Description** Displays a dialog box that allows the user to select from a list of choices and returns an Integer containing the index of the item that was selected.

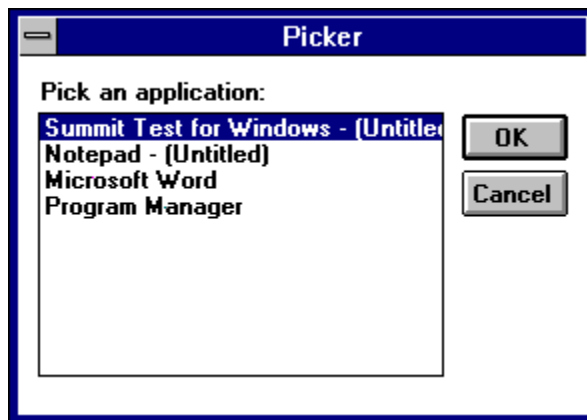
**Comments** The SelectBox statement accepts the following parameters:

| Parameter           | Description                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>title</i>        | Title of the dialog box. This can be an expression convertible to a String. A runtime error is generated if <i>title</i> is Null.                                                     |
| <i>prompt</i>       | Text to appear immediately above the list box containing the items. This can be an expression convertible to a String. A runtime error is generated if <i>prompt</i> is Null.         |
| <i>ArrayOfItems</i> | Single-dimensioned array. Each item from the array will occupy a single entry in the list box. A runtime error is generated if <i>ArrayOfItems</i> is not a single-dimensioned array. |

*ArrayOfItems* can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.

The value returned is an Integer representing the index of the item in the list box that was selected, with 0 being the first item. If the user selects Cancel, -1 is returned.

```
result% = SelectBox("Picker", "Pick an application:", a$)
```



**Example** 'This example gets the current apps running, puts them in to an array  
'and then asks the user to select one from a list.

```
Sub Main()
 Dim a$()
 AppList a$
 result% = SelectBox("Picker","Pick an application:",a$)
 If Not result% = -1 then
 MsgBox "User selected: " & a$(result%)
 Else
 MsgBox "User canceled"
 End If
End Sub
```

**See Also** MsgBox (statement); AskBox\$ (function); AskPassword\$ (function); InputBox,  
InputBox\$ (functions); OpenFilename\$ (function); SaveFilename\$ (function);  
AnswerBox (function).

**Platform(s)** Windows and Macintosh.

**Platform Notes:** Under Windows, SelectBox displays all text in its dialog box in 8-point MS  
Sans Serif.

**Windows**

---

## SelectButton (statement)

---

**Syntax** SelectButton *name\$* | *id*

**Description** Simulates a mouse click on the a push button given the push button's name (the  
*name\$* parameter) or ID (the *id* parameter).

**Comments** The SelectButton statement accepts the following parameters:

| Parameter                                                                                                                                                                                 | Description                                                                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name\$</i>                                                                                                                                                                             | String containing the name of the push button to be selected.                                                                                                                          |
| <i>id</i>                                                                                                                                                                                 | Integer representing the ID of the push button to be selected.<br><br>A runtime error is generated if a push button with the given name or ID cannot<br>be found in the active window. |
| <b>Note:</b> The SelectButton statement is used to select a button in another<br>application's dialog box. This command is not intended for use with built-in or<br>dynamic dialog boxes. |                                                                                                                                                                                        |

---

**Example** This example simulates the selection of several buttons in a dialog.

```
Sub Main()
 SelectButton "OK"
 SelectButton 2
 SelectButton "Close"
End Sub
```

**See Also** ButtonEnabled (function), ButtonExists (function)

**Platform(s)** Windows.

## SelectComboBoxItem (statement)

**Syntax** `SelectComboBoxItem {name$ | id}, {ItemName$ | ItemNumber} [, isDoubleClick]`

**Description** Selects an item from a combo box given the name or ID of the combo box and the name or line number of the item.

**Comments** The SelectComboBoxItem statement accepts the following parameters:

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                               |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name\$</i>        | String indicating the name of the combo box containing the item to be selected.<br><br>The name of a combo box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a combo box. A runtime error is generated if a combo box with that name cannot be found within the active window. |
| <i>id</i>            | Integer specifying the ID of the combo box containing the item to be selected.                                                                                                                                                                                                                                                                            |
| <i>ItemName\$</i>    | String specifying which item is to be selected. The string is compared without regard to case. If <i>ItemName\$</i> is a zero-length string, then all currently selected items are deselected. A runtime error results if <i>ItemName\$</i> cannot be found in the combo box.                                                                             |
| <i>ItemNumber</i>    | Integer containing the index of the item to be selected. A runtime error is generated if <i>ItemNumber</i> is not within the correct range.                                                                                                                                                                                                               |
| <i>isDoubleClick</i> | Boolean value indicating whether a double click of that item is to be simulated.                                                                                                                                                                                                                                                                          |

**Note:** The SelectComboBoxItem statement is used to set the item of a combo box in another application's dialog box. Use the DlgText statement to change the content of the text box part of a list box in a dynamic dialog box.

**Example** This example simulates the selection of a couple of comboboxes

```
Sub Main()
 SelectComboBoxItem "ComboBox1", "Item4"
 SelectComboBoxItem 1, 2, TRUE
End Sub
```

**See Also** ComboBoxEnabled (function); ComboBoxExists (function); GetComboBoxItem\$ (function); GetComboBoxItemCount (function).

**Platform(s)** Windows.

## SelectListBoxItem (statement)

**Syntax** `SelectListBoxItem {name$ | id}, {ItemName$ | ItemNumber} [, isDoubleClick]`

**Description** Selects an item from a list box given the name or ID of the list box and the name or line number of the item.

**Comments** The `SelectListBoxItem` statement accepts the following parameters:

| Parameter                                                                                                                                                                                                                                            | Description                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name\$</i>                                                                                                                                                                                                                                        | String indicating the name of the list box containing the item to be selected.<br><br>The name of a list box is determined by scanning the window list looking for a text control with the given name that is immediately followed by a list box. A runtime error is generated if a list box with that name cannot be found within the active window.                             |
| <i>id</i>                                                                                                                                                                                                                                            | Integer specifying the ID of the list box containing the item to be selected.                                                                                                                                                                                                                                                                                                     |
| <i>ItemName\$</i>                                                                                                                                                                                                                                    | String specifying which item is to be selected. The string is compared without regard to case. If <i>ItemName\$</i> is a zero-length string, then all currently selected items are deselected. A runtime error results if <i>ItemName\$</i> cannot be found in the list box.                                                                                                      |
| <i>ItemNumber</i>                                                                                                                                                                                                                                    | Integer containing the index of the item to be selected. A runtime error is generated if <i>ItemNumber</i> is not within the correct range.                                                                                                                                                                                                                                       |
| <i>isDoubleClick</i>                                                                                                                                                                                                                                 | Boolean value indicating whether a double click of that item is to be simulated.<br><br>The list box must exist within the current window or dialog box; otherwise, a runtime error will be generated.<br><br>For multiselect list boxes, <code>SelectListBoxItem</code> will select additional items (i.e., it will not remove the selection from the currently selected items). |
| <b>Note:</b> The <code>SelectListBoxItem</code> statement is used to select an item in a list box of another application's dialog box. Use the <code>DlgText</code> statement to change the selected item in a list box within a dynamic dialog box. |                                                                                                                                                                                                                                                                                                                                                                                   |

**Example** 'This example simulates a double click on the first item in list box 1.

```
Sub Main()
 SelectListBoxItem "ListBox1",1,TRUE
End Sub
```

**See Also** `GetListBoxItem$ (function)`; `GetListBoxItemCount (function)`;  
`ListBoxEnabled (function)`; `ListBoxExists (function)`.

**Platform(s)** Windows.

## SendKeys (statement)

**Syntax** `SendKeys KeyString$ [, [isWait] [, time]]`

**Description** Sends the specified keys to the active application, optionally waiting for the keys to be processed before continuing.

**Comments** The SendKeys statement accepts the following parameters:

| Parameter          | Description                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>KeyString\$</i> | String containing the keys to be sent. The format for <i>KeyString\$</i> is described below.                                                            |
| <i>isWait</i>      | Boolean value. If True (or not specified), then WM Basic waits for the keys to be completely processed before continuing.                               |
| <i>time</i>        | Integer specifying the number of milliseconds devoted for the output of the entire <i>KeyString\$</i> parameter. It must be within the following range: |

$$0 \leq time \leq 32767$$

For example, if *time* is 5000 (5 seconds) and the *KeyString\$* parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.

### Specifying Keys

To specify any key on the keyboard, simply use that key, such as "a" for lowercase a, or "A" for uppercase a.

Sequences of keys are specified by appending them together: "abc" or "dir /w".

Some keys have special meaning and are therefore specified in a special way—by enclosing them within braces. For example, to specify the percent sign, use "{%}". The following table shows the special keys:

| Key | Special Meaning              | Example                                         |
|-----|------------------------------|-------------------------------------------------|
| +   | Shift                        | " + {F1} "                      'Shift+F1       |
| ^   | Ctrl                         | " ^a "                          'Ctrl+A         |
| ~   | Shortcut for Enter           | " ~ "                          'Enter           |
| %   | Alt                          | " %F "                          'Alt+F          |
| [ ] | No special meaning           | " { [ } "                      'Open bracket    |
| { } | Used to enclose special keys | " {Up} "                      'Up Arrow         |
| ( ) | Used to specify grouping     | " ^ (ab) "                      'Ctrl+A, Ctrl+B |

Keys that are not displayed when you press them are also specified within braces, such as {Enter} or {Up}. A list of these keys follows:

|           |           |           |            |               |
|-----------|-----------|-----------|------------|---------------|
| {BkSp}    | {BS}      | {Break}   | {CapsLock} |               |
| {Clear}   |           |           |            |               |
| {Delete}  | {Del}     | {Down}    | {End}      | {Enter}       |
| {Escape}  | {Esc}     | {Help}    | {Home}     |               |
| {Insert}  |           |           |            |               |
| {Left}    | {NumLock} | {NumPad0} | {NumPad1}  | {NumPad2}     |
| {NumPad3} | {NumPad4} | {NumPad5} | {NumPad6}  | {NumPad7}     |
| {NumPad8} | {NumPad9} | {NumPad/} | {NumPad*}  | {NumPad-}     |
| {NumPad+} | {NumPad.} | {PgDn}    | {PgUp}     |               |
| {PrtSc}   |           |           |            |               |
| {Right}   | {Tab}     | {Up}      | {F1}       | {Scroll Lock} |
| {F2}      | {F3}      | {F4}      | {F5}       | {F6}          |
| {F7}      | {F8}      | {F9}      | {F10}      | {F11}         |
| {F12}     | {F13}     | {F14}     | {F15}      | {F16}         |

Keys can be combined with Shift, Ctrl, and Alt using the reserved keys "+", "^", and "%" respectively:

| For Key Combination | Use |
|---------------------|-----|
|---------------------|-----|

|             |               |
|-------------|---------------|
| Shift+Enter | " + {Enter} " |
| Ctrl+C      | " ^c "        |
| Alt+F2      | " % {F2} "    |

To specify a modifier key combined with a sequence of consecutive keys, group the key sequence within parentheses, as in the following example:

| For Key Combination | Use |
|---------------------|-----|
|---------------------|-----|

|                  |                     |
|------------------|---------------------|
| Shift+A, Shift+B | " + ( abc ) "       |
| Ctrl+F1, Ctrl+F2 | " ^ ( {F1} {F2} ) " |

Use "~" as a shortcut for embedding Enter within a key sequence:

| For Key Combination | Use |
|---------------------|-----|
|---------------------|-----|

|                   |           |
|-------------------|-----------|
| a, b, Enter, d, e | " ab~de " |
| Enter, Enter      | " ~ ~ "   |

To embed quotation marks, use two quotation marks in a row:

| For Key Combination | Use |
|---------------------|-----|
|---------------------|-----|

|         |             |
|---------|-------------|
| "Hello" | " "Hello" " |
| a"b"c   | "a" "b" "c" |

Key sequences can be repeated using a repeat count within braces:

| For Key Combination | Use |
|---------------------|-----|
|---------------------|-----|

|                |             |
|----------------|-------------|
| Ten "a" keys   | "{a 10}"    |
| Two Enter keys | "{Enter 2}" |

**Example** 'This example runs Notepad, writes to Notepad, and saves the new file using the SendKeys statement.

```
Sub Main()
 id = Shell("Notepad.exe")
 AppActivate "Notepad"
 SendKeys "Hello, Notepad." 'Write some text.
 Sleep 2000
 SendKeys "%fs" 'Save file (simulate Alt+F, S keys).
 Sleep 2000
 SendKeys "name.txt{ENTER}" 'Enter name of new file to save.
 AppClose "Notepad"
End Sub
```

**See Also** DoKeys (statement); QueKeys (statement); QueKeyDn (statement); QueKeyUp (statement).

**Platform(s)** Windows.

## Set (statement)

**Syntax 1** Set *object\_var* = *object\_expression*

**Syntax 2** Set *object\_var* = New *object\_type*

**Syntax 3** Set *object\_var* = Nothing

**Description** Assigns a value to an object variable.

**Comments**   **Syntax 1**

The first syntax assigns the result of an expression to an object variable. This statement does not duplicate the object being assigned but rather copies a reference of an existing object to an object variable.

The *object\_expression* is any expression that evaluates to an object of the same type as the *object\_var*.

With data objects, Set performs additional processing. When the Set is performed, the object is notified that a reference to it is being made and destroyed. For example, the following statement deletes a reference to object A, then adds a new reference to B.

```
Set a = b
```

In this way, an object that is no longer being referenced can be destroyed.

**Syntax 2**

In the second syntax, the object variable is being assigned to a new instance of an existing object type. This syntax is valid only for data objects.

When an object created using the New keyword goes out of scope (i.e., the Sub or Function in which the variable is declared ends), the object is destroyed.

**Syntax 3**

The reserved keyword Nothing is used to make an object variable reference no object. At a later time, the object variable can be compared to Nothing to test whether the object variable has been instantiated:

```
Set a = Nothing
:
If a Is Nothing Then Beep
```

**Example**   'This example creates two objects and sets their values.

```
Sub Main()
 Dim document As Object
 Dim page As Object
 Set document = GetObject("c:\resume.doc")
 Set page = Document.ActivePage
 MsgBox page.name
End Sub
```

**See Also**   = (statement); Let (statement); CreateObject (function); GetObject (function); Nothing (constant).

**Platform(s)**   Windows and Macintosh.

---

**SetAttr (statement)**

**Syntax**   SetAttr *filename\$,attribute*



**Description** Changes the attribute *filename\$* to the given attribute. A runtime error results if the file cannot be found.

**Comments** The SetAttr statement accepts the following parameters:

| Parameter | Description |
|-----------|-------------|
|-----------|-------------|

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>filename\$</i> | String containing the name of the file. |
|-------------------|-----------------------------------------|

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>attribute</i> | Integer specifying the new attribute of the file. |
|------------------|---------------------------------------------------|

The *attribute* parameter can contain any combination of the following values:

| Constant   | Value | Description                                   |
|------------|-------|-----------------------------------------------|
| ebNormal   | 0     | Turns off all attributes                      |
| ebReadOnly | 1     | Read-only files                               |
| ebHidden   | 2     | Hidden files                                  |
| ebSystem   | 4     | System files                                  |
| ebVolume   | 8     | Volume label                                  |
| ebArchive  | 32    | Files that have changed since the last backup |
| ebNone     | 64    | Turns off all attributes                      |

The attributes can be combined using the + operator or the binary Or operator.

**Example** 'This example creates a file and sets its attributes to Read-Only and 'System.

```
Sub Main()
 Open "test.dat" For Output Access Write As #1
 Close
 MsgBox "The current file attribute is: " & GetAttr("test.dat")
 SetAttr "test.dat", ebReadOnly Or ebSystem
 MsgBox "The file attribute was set to: " & GetAttr("test.dat")
End Sub
```

**See Also** GetAttr (function); FileAttr (function).

**Platform(s)** Windows and Macintosh.

**Platform** Under Windows, these attributes are the same as those used by DOS.

**Notes:**

**Windows**

## SetCheckBox (statement)

**Syntax** SetCheckBox {*name\$* | *id*}, *state*

**Description** Sets the state of the check box with the given name or ID.

**Comments** The `SetCheckBox` statement accepts the following parameters:

| Parameter                                                                                                                                                                                                                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name\$</i>                                                                                                                                                                                                                            | String containing the name of the check box to be set.                                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>id</i>                                                                                                                                                                                                                                | Integer specifying the ID of the check box to be set.                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>state</i>                                                                                                                                                                                                                             | Integer indicating the new state of the check box. If <i>state</i> is 1, then the box is checked. If <i>state</i> is 0, then the check is removed. If <i>state</i> is 2, then the box is dimmed (only applicable for three-state check boxes).<br><br>A runtime error is generated if a check box with the specified name cannot be found in the active window.<br><br>This statement has the side effect of setting the focus to the given check box. |
| <b>Note:</b> The <code>SetCheckBox</code> statement is used to set the state of a check box in another application's dialog box. Use the <code>DlgValue</code> statement to modify the state of a check box within a dynamic dialog box. |                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

**Example** 'This example sets a factious checkbox

```
Sub Main()
 SetCheckBox "CheckBox1",1
End Sub
```

**See Also** `CheckBoxExists` (function); `CheckBoxEnabled` (function); `GetCheckBox` (function); `DlgValue` (statement).

**Platform(s)** Windows.

## SetEditText (statement)

---

**Syntax** `SetEditText {name$ | id} ,content$`

**Description** Sets the content of an edit control given its name or ID.

**Comments** The SetEditText statement accepts the following parameters:

| Parameter        | Description                                                                                                                                                                                                                                                                                                                                                |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name\$</i>    | String containing the name of the text box to be set.<br><br>The name of a text box control is determined by scanning the window list looking for a text control with the given name that is immediately followed by an edit control. A runtime error is generated if a text box control with that name cannot be found within the active window.          |
| <i>id</i>        | Integer specifying the ID of the text box to be set.<br><br>For text boxes that do not have a preceding text control, the <i>id</i> can be used to absolutely reference the control. The <i>id</i> is determined by examining the dialog box with a resource editor or using an application such as Spy.                                                   |
| <i>content\$</i> | String containing the new content for the text box.<br><br>This statement has the side effect of setting the focus to the given text box.<br><br><b>Note:</b> The SetEditText statement is used to set the content of a text box in another application's dialog box. Use the DlgText statement to set the text of a text box within a dynamic dialog box. |

**Example** 'This example sets the content of the filename text box of the 'current window to "test.dat".

```
Sub Main()
 SetEditText "Filename:", "test.dat"
End Sub
```

**See Also** EditEnabled (function); EditExists (function); GetEditText\$ (function).

**Platform(s)** Windows.

## SetOption (statement)

**Syntax** SetOption *name\$* | *id*

**Description** Selects the specified option button given its name or ID.

**Comments** The `SetOption` statement accepts the following parameters:

| Parameter                                                                                                                                                                                                                 | Description                                                                                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name\$</i>                                                                                                                                                                                                             | String containing the name of the option button to be selected.                                                                                               |
| <i>id</i>                                                                                                                                                                                                                 | Integer containing the ID of the option button to be selected.<br>A runtime error is generated if the option button cannot be found within the active window. |
| <b>Note:</b> The <code>SetOption</code> statement is used to select an option button in another application's dialog box. Use the <code>DlgValue</code> statement to select an option button within a dynamic dialog box. |                                                                                                                                                               |

**Example** 'This example selects the Continue option button.

```
Sub Main()
 SetOption "Continue"
End Sub
```

**See Also** `GetOption` (function); `OptionEnabled` (function); `OptionExists` (function).

**Platform(s)** Windows.

## Sgn (function)

---

**Syntax** `Sgn(number)`

**Description** Returns an Integer indicating whether a number is less than, greater than, or equal to 0.

**Comments** Returns 1 if *number* is greater than 0.

Returns 0 if *number* is equal to 0.

Returns -1 if *number* is less than 0.

The *number* parameter is a numeric expression of any type. If *number* is `Null`, then a runtime error is generated. Empty is treated as 0.

**Example** 'This example tests the product of two numbers and displays  
'a message based on the sign of the result.

```
Sub Main()
 a% = -100
 b% = 100
 c% = a% * b%
 Select Case Sgn(c%)
 Case -1
 MsgBox "The product is negative " & Sgn(c%)
 Case 0
 MsgBox "The product is 0 " & Sgn(c%)
 Case 1
 MsgBox "The product is positive " & Sgn(c%)
 End Select
End Sub
```

**See Also** Abs (function).

**Platform(s)** Windows and Macintosh.

## Shell (function)

---

**Syntax** Shell(*command\$* [ , *WindowStyle* ])

**Description** Executes another application, returning the task ID if successful.

**Comments** The `Shell` statement accepts the following parameters:

| Parameter          | Description                                                                                                                                                                                                                                                                                                                                                                                                     |   |                          |   |                                |   |                      |   |                             |   |                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------------------|---|--------------------------------|---|----------------------|---|-----------------------------|---|-------------------------|
| <i>command\$</i>   | String containing the name of the application and any parameters.                                                                                                                                                                                                                                                                                                                                               |   |                          |   |                                |   |                      |   |                             |   |                         |
| <i>WindowStyle</i> | Optional Integer specifying the state of the application window after execution. It can be any of the following values: <table><tr><td>1</td><td>Normal window with focus</td></tr><tr><td>2</td><td>Minimized with focus (default)</td></tr><tr><td>3</td><td>Maximized with focus</td></tr><tr><td>4</td><td>Normal window without focus</td></tr><tr><td>7</td><td>Minimized without focus</td></tr></table> | 1 | Normal window with focus | 2 | Minimized with focus (default) | 3 | Maximized with focus | 4 | Normal window without focus | 7 | Minimized without focus |
| 1                  | Normal window with focus                                                                                                                                                                                                                                                                                                                                                                                        |   |                          |   |                                |   |                      |   |                             |   |                         |
| 2                  | Minimized with focus (default)                                                                                                                                                                                                                                                                                                                                                                                  |   |                          |   |                                |   |                      |   |                             |   |                         |
| 3                  | Maximized with focus                                                                                                                                                                                                                                                                                                                                                                                            |   |                          |   |                                |   |                      |   |                             |   |                         |
| 4                  | Normal window without focus                                                                                                                                                                                                                                                                                                                                                                                     |   |                          |   |                                |   |                      |   |                             |   |                         |
| 7                  | Minimized without focus                                                                                                                                                                                                                                                                                                                                                                                         |   |                          |   |                                |   |                      |   |                             |   |                         |

An error is generated if unsuccessful running *command\$*.

The `Shell` command runs programs asynchronously: the statement following the `Shell` statement will execute before the child application has exited. On some platforms, the next statement will run before the child application has finished loading.

The `Shell` function returns a value suitable for activating the application using the `AppActivate` statement. It is important that this value be placed into a Variant, as its type depends on the platform.

**Example** 'This example displays the Windows Clock, delays awhile, then closes it.

```
Sub Main()
 id = Shell("clock.exe",1)
 AppActivate "Clock"
 Sleep(2000)
 AppClose "Clock"
End Sub
```

**See Also** `PrintFile` (function); `SendKeys` (statement); `AppActivate` (statement).

**Platform(s)** Windows and Macintosh.

**Platform** The Macintosh does not support wildcard characters such as \* and ?. These are valid filename characters. Instead of wildcards, the Macintosh uses the `MacID` function to specify a collection of files of the same type. The syntax for this function is:

```
Shell(MacID(text$) [, WindowStyle])
```

The *text\$* parameter is a four-character string containing an application signature. A runtime error occurs if the `MacID` function is used on platforms other than the Macintosh.

On the Macintosh, the *WindowStyle* parameter only specifies whether the application receives the focus.

**Platform** Under Windows, this function returns the `hInstance` of the application.  
**Notes:** Since this value is only a WORD in size, the upper WORD of the result is always zero.  
**Windows**

## Sin (function)

**Syntax** `Sin(angle)`

**Description** Returns a `Double` value specifying the sine of *angle*.

**Comments** The *angle* parameter is a `Double` specifying an angle in radians.

**Example** 'This example displays the sine of pi/4 radians (45 degrees).

```
Sub Main()
 c# = Sin(Pi / 4)
 MsgBox "The sine of 45 degrees is: " & c#
End Sub
```

**See Also** `Tan (function)`; `Cos (function)`; `Atn (function)`.

**Platform(s)** Windows and Macintosh.

## Single (data type)

**Syntax** `Single`

**Description** A data type used to declare variables capable of holding real numbers with up to seven digits of precision.

## 444 Working Model Basic User's Manual

---

**Comments** Single variables are used to hold numbers within the following ranges:

| Sign               | Range                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Negative           | $-3.402823\text{E}38 \leq \text{single} \leq -1.401298\text{E}-45$                                                                                                                                                         |
| Positive           | $1.401298\text{E}-45 \leq \text{single} \leq 3.402823\text{E}38$                                                                                                                                                           |
|                    | The type-declaration character for Single is !.                                                                                                                                                                            |
|                    | <b>Storage</b>                                                                                                                                                                                                             |
|                    | Internally, singles are stored as 4-byte (32-bit) IEEE values. Thus, when appearing within a structure, singles require 4 bytes of storage. When used with binary or random files, 4 bytes of storage is required.         |
|                    | Each single consists of the following                                                                                                                                                                                      |
|                    | <ul style="list-style-type: none"><li>▪ A 1-bit sign</li><li>▪ An 8-bit exponent</li><li>▪ A 24-bit mantissa</li></ul>                                                                                                     |
| <b>See Also</b>    | Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); String (data type); Variant (data type); Boolean (data type); DefType (statement); CSng (function). |
| <b>Platform(s)</b> | Windows and Macintosh.                                                                                                                                                                                                     |

## Sleep (statement)

---

**Syntax** Sleep *milliseconds*

**Description** Causes the script to pause for a specified number of milliseconds.

**Comments** The *milliseconds* parameter is a Long in the following range:

$0 \leq \text{milliseconds} \leq 2,147,483,647$

**Example** 'This example displays a message for 2 seconds.

```
Sub Main()
 MsgOpen "Waiting 2 seconds",0,False,False
 Sleep(2000)
 MsgClose
End Sub
```

**Platform(s)** Windows and Macintosh.

**Platform** Under Windows, the accuracy of the system clock is modulo 55 milliseconds.

**Notes:** The value of *milliseconds* will, in the worst case, be rounded up to the nearest

**Windows** multiple of 55. In other words, if *milliseconds* is 1, it will be rounded to 55 in the worst case.



## Sln (function)

**Syntax** `Sln( Cost , Salvage , Life )`

**Description** Returns the straight-line depreciation of an asset assuming constant benefit from the asset.

**Comments** The Sln of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers.

The formula used to find the Sln of an asset is as follows:

$$(\text{Cost} - \text{Salvage Value}) / \text{Useful Life}$$

The Sln function requires the following parameters:

| Parameter      | Description                                                                         |
|----------------|-------------------------------------------------------------------------------------|
| <i>Cost</i>    | Double representing the initial cost of the asset.                                  |
| <i>Salvage</i> | Double representing the estimated value of the asset at the end of its useful life. |
| <i>Life</i>    | Double representing the length of the asset's useful life.                          |

The unit of time used to express the useful life of the asset is the same as the unit of time used to express the period for which the depreciation is returned.

**Example** 'This example calculates the straight-line depreciation of an asset that cost \$10,000.00 and has a salvage value of \$500.00 as scrap after 10 years of service life.

```
Sub Main()
 dep# = Sln(10000.00,500.00,10)
 MsgBox "The annual depreciation is: " & Format(dep#,"Currency")
End Sub
```

**See Also** SYD (function); DDB (function).

**Platform(s)** Windows and Macintosh.

## Space, Space\$ (functions)

**Syntax** `Space[$](NumSpaces)`

**Description** Returns a string containing the specified number of spaces.

**Comments** Space\$ returns a String, whereas Space returns a String variant.

*NumSpaces* is an Integer between 0 and 32767.

**Example** 'This example returns a string of ten spaces and displays it.

```
Sub Main()
 ln$ = Space$(10)
 MsgBox "Hello" & ln$ & "over there."
End Sub
```

**See Also** String, String\$ (functions); Spc (function).

**Platform(s)** Windows and Macintosh.

## Spc (function)

---

**Syntax** Spc(*numspaces*)

**Description** Prints out the specified number of spaces. This function can only be used with the Print and Print# statements.

**Comments** The *numspaces* parameter is an Integer specifying the number of spaces to be printed. It can be any value between 0 and 32767.

If a line width has been specified (using the Width statement), then the number of spaces is adjusted as follows:

```
numspaces = numspaces Mod width
```

If the resultant number of spaces is greater than width - print\_position, then the number of spaces is recalculated as follows:

```
numspaces = numspaces - (width - print_position)
```

These calculations have the effect of never allowing the spaces to overflow the line length. Furthermore, with a large value for column and a small line width, the file pointer will never advance more than one line.

**Example** 'This example displays 20 spaces between the arrows.

```
Sub Main()
 ViewportOpen
 Print "I am"; Spc(20); "20 spaces apart!"
 Sleep (10000)'Wait 10 seconds.
 ViewportClose
End Sub
```

**See Also** Tab (function); Print (statement); Print# (statement).

**Platform(s)** Windows and Macintosh.

## SQLBind (function)

---

**Syntax** SQLBind(*ID*,*array*,*column*)

**Description** Specifies which fields are returned when results are requested using the SQLRetrieve or SQLRetrieveToFile function.

**Comments** The following table describes the parameters to the SQLBind function:

| Parameter | Description |
|-----------|-------------|
|-----------|-------------|

|           |                                               |
|-----------|-----------------------------------------------|
| <i>ID</i> | Long parameter specifying a valid connection. |
|-----------|-----------------------------------------------|

|              |                                                                                                                                                                                                                                                                   |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>array</i> | Any array of variants. Each call to SQLBind adds a new column number (an Integer) in the appropriate slot in the array. Thus, as you bind additional columns, the <i>array</i> parameter grows, accumulating a sorted list (in ascending order) of bound columns. |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

If *array* is fixed, then it must be a one-dimensional variant array with sufficient space to hold all the bound column numbers. A runtime error is generated if *array* is too small.

If *array* is dynamic, then it will be resized to exactly hold all the bound column numbers.

|               |                                                                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>column</i> | Optional Long parameter that specifies the column to which to bind data. If this parameter is omitted, all bindings for the connection are dropped. |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|

This function returns the number of bound columns on the connection. If no columns are bound, then 0 is returned. If there are no pending queries, then calling SQLBind will cause an error (queries are initiated using the SQLExecQuery function).

If supported by the driver, row numbers can be returned by binding column 0.

WM Basic generates a trappable runtime error if SQLBind fails. Additional error information can then be retrieved using the SQLError function.

**Example** 'This example binds columns to data.

```
Sub Main()
 Dim columns() As Variant
 id& = SQLOpen("dsn=SAMPLE", , 3)
 t& = SQLExecQuery(id&, "Select * From c:\sample.dbf")
 i% = SQLBind(id&, columns, 3)
 i% = SQLBind(id&, columns, 1)
 i% = SQLBind(id&, columns, 2)
 i% = SQLBind(id&, columns, 6)
 For x = 0 To (i% - 1)
 MsgBox columns(x)
 Next x
 id& = SQLClose(id&)
End Sub
```

**See Also** SQLRetrieve (function); SQLRetrieveToFile (function).

**Platform(s)** Windows.

## SQLClose (function)

---

**Syntax** `SQLClose(connectionID)`

**Description** Closes the connection to the specified data source.

**Comments** The unique connection ID (*connectionID*) is a Long value representing a valid connection as returned by `SQLOpen`. After `SQLClose` is called, any subsequent calls made with the *connectionID* will generate runtime errors.

The `SQLClose` function returns 0 if successful; otherwise, it returns the passed connection ID and generates a trappable runtime error. Additional error information can then be retrieved using the `SQLError` function.

WM Basic automatically closes all open SQL connections when either the script or the application terminates. You should use the `SQLClose` function rather than relying on WM Basic to automatically close connections in order to ensure that your connections are closed at the proper time.

**Example** 'This example disconnects the the data source sample.

```
Sub Main()
 id& = SQLOpen("dsn=SAMPLE",,3)
 id& = SQLClose(id&)
End Sub
```

**See Also** `SQLOpen` (function).

**Platform(s)** Windows.

## SQLError (function)

---

**Syntax** `SQLError(ErrArray [, ID])`

**Description** Retrieves driver-specific error information for the most recent SQL functions that failed.

**Comments** This function is called after any other SQL function fails. Error information is returned in a two-dimensional array (*ErrArray*). The following table describes the parameters to the `SQLError` function:

| Parameter           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ErrArray</i>     | Two-dimensional <code>Variant</code> array, which can be dynamic or fixed.<br><br>If the array is fixed, it must be ( <i>x</i> ,3), where <i>x</i> is the number of errors you want returned. If <i>x</i> is too small to hold all the errors, then the extra error information is discarded. If <i>x</i> is greater than the number of errors available, all errors are returned, and the empty array elements are set to <code>Empty</code> .<br><br>If the array is dynamic, it will be resized to hold the exact number of errors. |
| <i>ID</i>           | Optional <code>Long</code> parameter specifying a connection ID. If this parameter is omitted, error information is returned for the most recent SQL function call.<br><br>Each array entry in the <i>ErrArray</i> parameter describes one error. The three elements in each array entry contain the following information:                                                                                                                                                                                                            |
| Element             | Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| ( <i>entry</i> , 0) | The ODBC error state, indicated by a <code>Long</code> containing the error class and subclass.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ( <i>entry</i> , 1) | The ODBC native error code, indicated by a <code>Long</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| ( <i>entry</i> , 2) | The text error message returned by the driver. This field is <code>String</code> type.<br><br>For example, to retrieve the ODBC text error message of the first returned error, the array is referenced as:<br><br><code>ErrArray(0, 2)</code><br><br>The <code>SQLError</code> function returns the number of errors found.<br><br>WM Basic generates a runtime error if <code>SQLError</code> fails. (You cannot use the <code>SQLError</code> function to gather additional error information in this case.)                        |

**Example** 'This example forces a connection error and traps it for use with  
'the `SQL_Error` function.

```
Sub Main()
 Dim a() As Variant
 On Error Goto Trap
 id& = SQLOpen("",,4)
 id& = SQLClose(id&)
 Exit Sub

Trap:
 rc% = SQL_Error(a)
 If (rc%) Then
 For x = 0 To (rc% - 1)
 MsgBox "The SQLState returned was: " & a(x,0)
 MsgBox "The native error code returned was: " & a(x,1)
 MsgBox a(x,2)
 Next x
 End If
End Sub
```

**Platform(s)** Windows.

---

## SQLExecQuery (function)

---

**Syntax** `SQLExecQuery(ID, query$)`

**Description** Executes an SQL statement query on a data source.

**Comments** This function is called after a connection to a data source is established using the `SQLOpen` function. The `SQLExecQuery` function may be called multiple times with the same connection ID, each time replacing all results.

The following table describes the parameters to the `SQLExecQuery` function:

| Parameter       | Description                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------------------|
| <i>ID</i>       | Long identifying a valid connected data source. This parameter is returned by the <code>SQLOpen</code> function. |
| <i>query</i> \$ | String specifying an SQL query statement. The SQL syntax of the string must strictly follow that of the driver.  |

The return value of this function depends on the result returned by the SQL statement:

| SQL Statement          | Value                                                                      |
|------------------------|----------------------------------------------------------------------------|
| SELECT...FROM          | The value returned is the number of columns returned by the SQL statement. |
| DELETE, INSERT, UPDATE | The value returned is the number of rows affected by the SQL statement.    |

WM Basic generates a runtime error if `SQLExecQuery` fails. Additional error information can then be retrieved using the `SQLError` function.

**Example** 'This example executes a query on the connected data source.

```
Sub Main()
 Dim s As String
 Dim qry As Long
 Dim a() As Variant

 On Error Goto Trap
 id& = SQLOpen("dsn=SAMPLE", s$, 3)
 qry = SQLExecQuery(id&,"Select * From c:\sample.dbf")
 MsgBox "There are " & qry & " columns in the result set."
 id& = SQLClose(id&)
 Exit Sub

Trap:
 rc% = SQLError(a)
 If (rc%) Then
 For x = 0 To (rc% - 1)
 MsgBox "The SQLState returned was: " & a(x,0)
 MsgBox "The native error code returned was: " & a(x,1)
 MsgBox a(x,2)
 Next x
 End If
End Sub
```

**See Also** `SQLOpen (function)`; `SQLClose (function)`; `SQLRetrieve (function)`; `SQLRetrieveToFile (function)`.

**Platform(s)** Windows.

## SQLGetSchema (function)

**Syntax** `SQLGetSchema (ID, action, [ , [array] [ , qualifier$] ])`

**Description** Returns information about the data source associated with the specified connection.

**Comments** The following table describes the parameters to the `SQLGetSchema` function:

| Parameter      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|--------------------|-------|-----------------------------------------------------------------------------------------------------------------|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|--------------------|----------------|-------------------------|----------------|--------------------|-----------|-------------------|-------|--------------------|-----------|--|---|---|----------------|-------------------------|----------------|--------------------|-----------|--|---|---------------------------------------------------------|
| <i>ID</i>      | Long parameter identifying a valid connected data source. This parameter is returned by the <code>SQLOpen</code> function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| <i>action</i>  | Integer parameter specifying the results to be returned. The following table lists values for this parameter:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
|                | <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>1</td><td>Returns a one-dimensional array of available data sources. The array is returned in the <i>array</i> parameter.</td></tr> <tr> <td>2</td><td>Returns a one-dimensional array of databases (either directory names or database names, depending on the driver) associated with the current connection. The array is returned in the <i>array</i> parameter.</td></tr> <tr> <td>3</td><td>Returns a one-dimensional array of owners (user IDs) of the database associated with the current connection. The array is returned in the <i>array</i> parameter.</td></tr> <tr> <td>4</td><td>Returns a one-dimensional array of table names for a specified owner and database associated with the current connection. The array is returned in the <i>array</i> parameter.</td></tr> <tr> <td>5</td><td> <p>Returns a two-dimensional array (<i>n</i> by 2) containing information about a specified table. The array is configured as follows:</p> <table> <tr> <td>(0,0)</td><td>Zeroth column name</td></tr> <tr> <td>(0,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> <tr> <td>(1,0)</td><td>First column name</td></tr> <tr> <td>(1,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> <tr> <td>:</td><td>:</td></tr> <tr> <td>(<i>n</i>,0)</td><td><i>N</i>th column name</td></tr> <tr> <td>(<i>n</i>,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> </table> </td></tr> <tr> <td>6</td><td>Returns a string containing the ID of the current user.</td></tr> </table> | Value | Meaning            | 1     | Returns a one-dimensional array of available data sources. The array is returned in the <i>array</i> parameter. | 2         | Returns a one-dimensional array of databases (either directory names or database names, depending on the driver) associated with the current connection. The array is returned in the <i>array</i> parameter. | 3     | Returns a one-dimensional array of owners (user IDs) of the database associated with the current connection. The array is returned in the <i>array</i> parameter. | 4     | Returns a one-dimensional array of table names for a specified owner and database associated with the current connection. The array is returned in the <i>array</i> parameter. | 5         | <p>Returns a two-dimensional array (<i>n</i> by 2) containing information about a specified table. The array is configured as follows:</p> <table> <tr> <td>(0,0)</td><td>Zeroth column name</td></tr> <tr> <td>(0,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> <tr> <td>(1,0)</td><td>First column name</td></tr> <tr> <td>(1,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> <tr> <td>:</td><td>:</td></tr> <tr> <td>(<i>n</i>,0)</td><td><i>N</i>th column name</td></tr> <tr> <td>(<i>n</i>,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> </table> | (0,0) | Zeroth column name | (0,1)          | ODBC SQL data type      | (Integer)      |                    | (1,0)     | First column name | (1,1) | ODBC SQL data type | (Integer) |  | : | : | ( <i>n</i> ,0) | <i>N</i> th column name | ( <i>n</i> ,1) | ODBC SQL data type | (Integer) |  | 6 | Returns a string containing the ID of the current user. |
| Value          | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| 1              | Returns a one-dimensional array of available data sources. The array is returned in the <i>array</i> parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| 2              | Returns a one-dimensional array of databases (either directory names or database names, depending on the driver) associated with the current connection. The array is returned in the <i>array</i> parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| 3              | Returns a one-dimensional array of owners (user IDs) of the database associated with the current connection. The array is returned in the <i>array</i> parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| 4              | Returns a one-dimensional array of table names for a specified owner and database associated with the current connection. The array is returned in the <i>array</i> parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| 5              | <p>Returns a two-dimensional array (<i>n</i> by 2) containing information about a specified table. The array is configured as follows:</p> <table> <tr> <td>(0,0)</td><td>Zeroth column name</td></tr> <tr> <td>(0,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> <tr> <td>(1,0)</td><td>First column name</td></tr> <tr> <td>(1,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> <tr> <td>:</td><td>:</td></tr> <tr> <td>(<i>n</i>,0)</td><td><i>N</i>th column name</td></tr> <tr> <td>(<i>n</i>,1)</td><td>ODBC SQL data type</td></tr> <tr> <td>(Integer)</td><td></td></tr> </table>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | (0,0) | Zeroth column name | (0,1) | ODBC SQL data type                                                                                              | (Integer) |                                                                                                                                                                                                               | (1,0) | First column name                                                                                                                                                 | (1,1) | ODBC SQL data type                                                                                                                                                             | (Integer) |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | :     | :                  | ( <i>n</i> ,0) | <i>N</i> th column name | ( <i>n</i> ,1) | ODBC SQL data type | (Integer) |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| (0,0)          | Zeroth column name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| (0,1)          | ODBC SQL data type                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| (Integer)      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| (1,0)          | First column name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| (1,1)          | ODBC SQL data type                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| (Integer)      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| :              | :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| ( <i>n</i> ,0) | <i>N</i> th column name                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| ( <i>n</i> ,1) | ODBC SQL data type                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| (Integer)      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |
| 6              | Returns a string containing the ID of the current user.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |                    |       |                                                                                                                 |           |                                                                                                                                                                                                               |       |                                                                                                                                                                   |       |                                                                                                                                                                                |           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |                    |                |                         |                |                    |           |                   |       |                    |           |  |   |   |                |                         |                |                    |           |  |   |                                                         |



- |    |                                                                                                                                            |
|----|--------------------------------------------------------------------------------------------------------------------------------------------|
| 7  | Returns a string containing the name (either the directory name or the database name, depending on the driver) of the current database.    |
| 8  | Returns a string containing the name of the data source on the current connection.                                                         |
| 9  | Returns a string containing the name of the DBMS of the data source on the current connection (e.g., "FoxPro 2.5" or "Excel Files").       |
| 10 | Returns a string containing the name of the server for the data source.                                                                    |
| 11 | Returns a string containing the owner qualifier used by the data source (e.g., "owner," "Authorization ID," "Schema").                     |
| 12 | Returns a string containing the table qualifier used by the data source (e.g., "table," "file").                                           |
| 13 | Returns a string containing the database qualifier used by the data source (e.g., "database," "directory").                                |
| 14 | Returns a string containing the procedure qualifier used by the data source (e.g., "database procedure," "stored procedure," "procedure"). |

*array* Optional Variant array parameter. This parameter is only required for action values 1, 2, 3, 4, and 5. The returned information is put into this array.

If *array* is fixed and it is not the correct size necessary to hold the requested information, then `SQLGetSchema` will fail. If the array is larger than required, then any additional elements are erased.

If *array* is dynamic, then it will be redimensioned to hold the exact number of elements requested.

*qualifier*

Optional `String` parameter required for actions 3, 4, or 5. The values are listed in the following table:

| Action | Qualifier                                                                                                                                       |
|--------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| 3      | The <i>qualifier</i> parameter must be the name of the database represented by <i>ID</i> .                                                      |
| 4      | The <i>qualifier</i> parameter specifies a database name and an owner name. The syntax for this string is:<br><br><i>DatabaseName.OwnerName</i> |
| 5      | The <i>qualifier</i> parameter specifies the name of a table on the current connection.                                                         |

WM Basic generates a runtime error if `SQLGetSchema` fails. Additional error information can then be retrieved using the `SQLError` function.

If you want to retrieve the available data sources (where *action* = 1) before establishing a connection, you can pass 0 as the *ID* parameter. This is the only action that will execute successfully without a valid connection.

This function calls the ODBC functions `SQLGetInfo` and `SQLTables` in order to retrieve the requested information. Some database drivers do not support these calls and will therefore cause the `SQLGetSchema` function to fail.

**Example** 'This example gets all available data sources.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 Dim dsn() As Variant
 numdims% = SQLGetSchema(0,1,dsn)
 If (numdims%) Then
 msg = "Valid data sources are:" & crlf
 For x = 0 To numdims% - 1
 msg = msg & dsn(x) & crlf
 Next x
 Else
 msg = "There are no available data sources."
 End If
 MsgBox msg
End Sub
```

**See Also** `SQLOpen` (function).

**Platform(s)** Windows.

## SQLOpen (function)

**Syntax** `SQLOpen(login$ [, [completed$] [, prompt]])`

**Description** Establishes a connection to the specified data source, returning a `Long` representing the unique connection ID.

**Comments** This function connects to a data source using a login string (*login\$*) and optionally sets the completed login string (*completed\$*) that was used by the driver. The following table describes the parameters to the `SQLOpen` function:

| Parameter          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------|---|----------------------------------------------------|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---------------------------------------------------|
| <i>login\$</i>     | <code>String</code> expression containing information required by the driver to connect to the requested data source. The syntax must strictly follow the driver's SQL syntax.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |
| <i>completed\$</i> | Optional <code>String</code> variable that will receive a completed connection string returned by the driver. If this parameter is missing, then no connection string will be returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |
| <i>prompt</i>      | Integer expression specifying any of the following values: <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>1</td><td>The driver's login dialog box is always displayed.</td></tr> <tr> <td>2</td><td>The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. This is the default behavior.</td></tr> <tr> <td>3</td><td>The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. Dialog box options that were passed as valid parameters are dimmed and unavailable.</td></tr> <tr> <td>4</td><td>The driver's login dialog box is never displayed.</td></tr> </table> | Value | Meaning | 1 | The driver's login dialog box is always displayed. | 2 | The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. This is the default behavior. | 3 | The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. Dialog box options that were passed as valid parameters are dimmed and unavailable. | 4 | The driver's login dialog box is never displayed. |
| Value              | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |
| 1                  | The driver's login dialog box is always displayed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |
| 2                  | The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. This is the default behavior.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |
| 3                  | The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. Dialog box options that were passed as valid parameters are dimmed and unavailable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |
| 4                  | The driver's login dialog box is never displayed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |       |         |   |                                                    |   |                                                                                                                                                              |   |                                                                                                                                                                                                                    |   |                                                   |

The `SQLOpen` function will never return an invalid connection ID. The following example establishes a connection using the driver's login dialog box:

```
id& = SQLOpen(" ", 1)
```

WM Basic returns 0 and generates a trappable runtime error if `SQLOpen` fails. Additional error information can then be retrieved using the `SQLError` function.

Before you can use any SQL statements, you must set up a data source and relate an existing database to it. This is accomplished using the odbcadm.exe program.

**Example** 'This example connects the data source called "sample," returning the 'completed connection string, and then displays it.

```
Sub Main()
 Dim s As String
 id& = SQLOpen("dsn=SAMPLE",s$,3)
 MsgBox "The completed connection string is: " & s$
 id& = SQLClose(id&)
End Sub
```

**See Also** **SQLClose** (function).

**Platform(s)** Windows.

---

## SQLRequest (function)

---

**Syntax** **SQLRequest**(*connection\$*,*query\$*,*array* [, [*output\$*] [, [*prompt*] [, [*isColumnNames*]]])

**Description** Opens a connection, runs a query, and returns the results as an array.

**Comments** The **SQLRequest** function takes the following parameters:

| Parameter            | Description                                                                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>connection</i>    | String specifying the connection information required to connect to the data source.                                                                                                                                                                                              |
| <i>query</i>         | String specifying the query to execute. The syntax of this string must strictly follow the syntax of the ODBC driver.                                                                                                                                                             |
| <i>array</i>         | Array of variants to be filled with the results of the query.<br><br>The <i>array</i> parameter must be dynamic: it will be resized to hold the exact number of records and fields.                                                                                               |
| <i>output</i>        | Optional String to receive the completed connection string as returned by the driver.                                                                                                                                                                                             |
| <i>prompt</i>        | Optional Integer specifying the behavior of the driver's dialog box.                                                                                                                                                                                                              |
| <i>isColumnNames</i> | Optional Boolean specifying whether the column names are returned as the first row of results. The default is False.<br><br>WM Basic generates a runtime error if <b>SQLRequest</b> fails. Additional error information can then be retrieved using the <b>SQLError</b> function. |

The `SQLRequest` function performs one of the following actions, depending on the type of query being performed:

| Type of Query          | Action                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SELECT                 | <p>The <code>SQLRequest</code> function fills <i>array</i> with the results of the query, returning a <code>Long</code> containing the number of results placed in the array. The array is filled as follows (assuming an <i>x</i> by <i>y</i> query):</p> <pre>(record 1,field 1) (record 1,field 2) : (record 1,field y) (record 2,field 1) (record 2,field 2) : (record 2,field y) : : (record x,field 1) (record x,field 2) : (record x,field y)</pre> |
| INSERT, DELETE, UPDATE | <p>The <code>SQLRequest</code> function erases <i>array</i> and returns a <code>Long</code> containing the number of affected rows.</p>                                                                                                                                                                                                                                                                                                                    |

**Example** 'This example opens a data source, runs a select query on it, and 'then displays all the data found in the result set.

```
Sub Main()
 Dim a() As Variant
 l& = SQLRequest("dsn=SAMPLE;", "Select * From
c:\sample.dbf", a, , 3, True)
 For x = 0 To Ubound(a)
 For y = 0 To l - 1
 MsgBox a(x,y)
 Next y
 Next x
End Sub
```

**Platform(s)** Windows.

## SQLRetrieve (function)

**Syntax** `SQLRetrieve(ID,array[, [maxcolumns] [, [ maxrows] [, [isColumnNames] [, isFetchFirst]]])`

**Description** Retrieves the results of a query.

**Comments** This function is called after a connection to a data source is established, a query is executed, and the desired columns are bound. The following table describes the parameters to the `SQLRetrieve` function:

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ID</i>            | Long identifying a valid connected data source with pending query results.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <i>array</i>         | Two-dimensional array of variants to receive the results. The array has <i>x</i> rows by <i>y</i> columns. The number of columns is determined by the number of bindings on the connection.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>maxcolumns</i>    | Optional Integer expression specifying the maximum number of columns to be returned. If <i>maxcolumns</i> is greater than the number of columns bound, the additional columns are set to empty. If <i>maxcolumns</i> is less than the number of bound results, the rightmost result columns are discarded until the result fits.                                                                                                                                                                                                                                                                               |
| <i>maxrows</i>       | Optional Integer specifying the maximum number of rows to be returned. If <i>maxrows</i> is greater than the number of rows available, all results are returned, and additional rows are set to empty. If <i>maxrows</i> is less than the number of rows available, the array is filled, and additional results are placed in memory for subsequent calls to <code>SQLRetrieve</code> .                                                                                                                                                                                                                        |
| <i>isColumnNames</i> | Optional Boolean specifying whether column names should be returned as the first row of results. The default is <code>False</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>isFetchFirst</i>  | <p>Optional Boolean expression specifying whether results are retrieved from the beginning of the result set. The default is <code>False</code>.</p> <p>Before you can retrieve the results from a query, you must (1) initiate a query by calling the <code>SQLExecQuery</code> function and (2) specify the fields to retrieve by calling the <code>SQLBind</code> function.</p> <p>This function returns a Long specifying the number of columns available in the array.</p> <p>WM Basic generates a runtime error if <code>SQLRetrieve</code> fails. Additional error information is placed in memory.</p> |

**Example** 'This example executes a query on the connected data source, binds 'columns, and retrieves them.

```
Sub Main()
 Dim a() As Variant
 Dim b() As Variant
 Dim c() As Variant

 On Error Goto Trap
 id& = SQLOpen("DSN=SAMPLE",,3)
 qry& = SQLExecQuery(id&,"Select * From c:\sample.dbf")
 i% = SQLBind(id&,b,3)
 i% = SQLBind(id&,b,1)
 i% = SQLBind(id&,b,2)
 i% = SQLBind(id&,b,6)
 l& = SQLRetrieve(id&,c)
 For x = 0 To Ubound(c)
 For y = 0 To l& - 1
 MsgBox c(x,y)
 Next y
 Next x
 id& = SQLClose(id&)
 Exit Sub

Trap:
 rc% = SQLError(a)
 If (rc%) Then
 For x = 0 To (rc% - 1)
 MsgBox "The SQLState returned was: " & a(x,0)
 MsgBox "The native error code returned was: " & a(x,1)
 MsgBox a(x,2)
 Next x
 End If
End Sub
```

**See Also** SQLOpen (function); SQLExecQuery (function); SQLClose (function); SQLBind (function); SQLRetrieveToFile (function).

**Platform(s)** Windows.

## SQLRetrieveToFile (function)

**Syntax** SQLRetrieveToFile(*ID*,*destination\$* [, [*isColumnNames*] [, *delimiter\$*]])

**Description** Retrieves the results of a query and writes them to the specified file.

**Comments** The following table describes the parameters to the `SQLRetrieveToFile` function:

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ID</i>            | Long specifying a valid connection ID.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>destination</i>   | String specifying the file where the results are written.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>isColumnNames</i> | Optional <code>Boolean</code> specifying whether the first row of results returned are the bound column names. By default, the column names are not returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>delimiter</i>     | <p>Optional <code>String</code> specifying the column separator. A tab (<code>Chr\$(9)</code>) is used as the default.</p> <p>Before you can retrieve the results from a query, you must (1) initiate a query by calling the <code>SQLExecQuery</code> function and (2) specify the fields to retrieve by calling the <code>SQLBind</code> function.</p> <p>This function returns the number of rows written to the file. A runtime error is generated if there are no pending results or if WM Basic is unable to open the specified file.</p> <p>WM Basic generates a runtime error if <code>SQLRetrieveToFile</code> fails. Additional error information may be placed in memory for later use with the <code>SQLError</code> function.</p> |



**Example** 'This example opens a connection, runs a query, binds columns, and writes the results to a file.

```
Sub Main()
 Dim a() As Variant
 Dim b() As Variant

 On Error Goto Trap
 id& = SQLOpen("DSN=SAMPLE;UID=RICH",,4)
 t& = SQLExecQuery(id&, "Select * From c:\sample.dbf")
 i% = SQLBind(id&,b,3)
 i% = SQLBind(id&,b,1)
 i% = SQLBind(id&,b,2)
 i% = SQLBind(id&,b,6)
 l& = SQLRetrieveToFile(id&,"c:\results.txt",True,"")
 id& = SQLClose(id&)
 Exit Sub

Trap:
 rc% = SQLError(a)
 If (rc%) Then
 For x = 0 To (rc-1)
 MsgBox "The SQLState returned was: " & a(x,0)
 MsgBox "The native error code returned was: " & a(x,1)
 MsgBox a(x,2)
 Next x
 End If
End Sub
```

**See Also** SQLOpen (function); SQLExecQuery (function); SQLClose (function); SQLBind (function); SQLRetrieve (function).

**Platform(s)** Windows.

## Sqr (function)

**Syntax** *Sqr(number)*

**Description** Returns a Double representing the square root of *number*.

**Comments** The *number* parameter is a Double greater than or equal to 0.

**Example** 'This example calculates the square root of the numbers from 1 to 10 and displays them.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 For x = 1 To 10
 sx# = Sqr(x)
 msg = msg & Format(x,"Fixed") & " - " & Format(sx#,"Fixed") &
crlf
 Next x
 MsgBox msg
End Sub
```

**Platform(s)** Windows and Macintosh.

## Stop (statement)

---

**Syntax** `Stop`

**Description** Suspends execution of the current script, returning control to a debugger if one is present. If a debugger is not present, this command will have the same effect as `End`.

**Example** 'The `Stop` statement can be used for debugging. In this example, it is used  
'to stop execution when `Z` is randomly set to 0.

```
Sub Main()
 For x = 1 To 10
 z = Random(0,10)
 If z = 0 Then Stop
 y = x / z
 Next x
End Sub
```

**See Also** `Exit For (statement); Exit Do (statement); Exit Function (statement); Exit Sub (statement); End (statement).`

**Platform(s)** Windows and Macintosh.

## Str, Str\$ (functions)

---

**Syntax** `Str[$](number)`

**Description** Returns a string representation of the given number.

**Comments** The *number* parameter is any numeric expression or expression convertible to a number. If *number* is negative, then the returned string will contain a leading minus sign. If *number* is positive, then the returned string will contain a leading space.

Singles are printed using only 7 significant digits. Doubles are printed using 15–16 significant digits.

These functions only output the period as the decimal separator and do not output thousands separators. Use the `CStr`, `Format`, or `Format$` function for this purpose.

**Example** 'In this example, the `Str$` function is used to display the value of a  
'numeric variable.

```
Sub Main()
 x# = 100.22
 MsgBox "The string value is: " + Str(x#)
End Sub
```

**See Also** Format, Format\$ (functions); CStr (function).

**Platform(s)** Windows and Macintosh.

## StrComp (function)

**Syntax** StrComp(*string1*, *string2* [, *compare*])

**Description** Returns an Integer indicating the result of comparing the two string arguments.

**Comments** Any of the following values are returned:

|      |                                          |
|------|------------------------------------------|
| 0    | <i>string1</i> = <i>string2</i>          |
| 1    | <i>string1</i> > <i>string2</i>          |
| -1   | <i>string1</i> < <i>string2</i>          |
| Null | <i>string1</i> or <i>string2</i> is Null |

The StrComp function accepts the following parameters:

| Parameter      | Description                                                                                                                                                                                                                                   |   |                           |   |                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---------------------------|---|-----------------------------|
| <i>string1</i> | First string to be compared, which can be any expression convertible to a String.                                                                                                                                                             |   |                           |   |                             |
| <i>string2</i> | Second string to be compared, which can be any expression convertible to a String.                                                                                                                                                            |   |                           |   |                             |
| <i>compare</i> | Optional Integer specifying how the comparison is to be performed. It can be either of the following values: <table> <tr> <td>0</td><td>Case-sensitive comparison</td></tr> <tr> <td>1</td><td>Case-insensitive comparison</td></tr> </table> | 0 | Case-sensitive comparison | 1 | Case-insensitive comparison |
| 0              | Case-sensitive comparison                                                                                                                                                                                                                     |   |                           |   |                             |
| 1              | Case-insensitive comparison                                                                                                                                                                                                                   |   |                           |   |                             |

If *compare* is not specified, then the current Option Compare setting is used. If no Option Compare statement has been encountered, then Binary is used (i.e., string comparison is case-sensitive).

**Example** 'This example compares two strings and displays the results.  
'It illustrates that the function compares two strings to the  
'length of the shorter string in determining equivalency.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 a$ = "This string is UPPERCASE and lowercase"
 b$ = "This string is uppercase and lowercase"
 c$ = "This string"
 d$ = "This string is uppercase and lowercase characters"
 abc = StrComp(a$,b$,0)
 msg = msg & "a and c (sensitive) : " & Format(abc,"True/False") &
 crlf
 abi = StrComp(a$,b$,1)
 msg = msg & "a and b (insensitive): " & Format(abi,"True/False") &
 crlf
 aci = StrComp(a$,c$,1)
 msg = msg & "a and c (insensitive): " & Format(aci,"True/False") &
 crlf
 bdi = StrComp(b$,d$,1)
 msg = msg & "b and d (sensitive) : " & Format(bdi,"True/False") &
 crlf
 MsgBox msg
End Sub
```

**See Also** Comparison Operators (topic); Like (operator); Option Compare (statement).

**Platform(s)** Windows and Macintosh.

---

## String (data type)

---

**Syntax** String

**Description** A data type capable of holding a number of characters.

**Comments** Strings are used to hold sequences of characters, each character having a value between 0 and 255. Strings can be any length up to a maximum length of 32767 characters.

Strings can contain embedded nulls, as shown in the following example:

```
s$ = "Hello" + Chr$(0) + "there" 'String with embedded null
```

The length of a string can be determined using the Len function. This function returns the number of characters that have been stored in the string, including unprintable characters.

The type-declaration character for String is \$.

String variables that have not yet been assigned are set to zero-length by default.

Strings are normally declared as variable-length, meaning that the memory required for storage of the string depends on the size of its content. The following WM Basic statements declare a variable-length string and assign it a value of length 5:

```
Dim s As String
s = "Hello" 'String has length 5.
```

Fixed-length strings are given a length in their declaration:

```
Dim s As String * 20
s = "Hello" 'String has length 20 (internally pads with
spaces).
```

When a string expression is assigned to a fixed-length string, the following rules apply:

- If the string expression is less than the length of the fixed-length string, then the fixed-length string is padded with spaces up to its declared length.
- If the string expression is greater than the length of the fixed-length string, then the string expression is truncated to the length of the fixed-length string.

Fixed-length strings are useful within structures when a fixed size is required, such as when passing structures to external routines.

The storage for a fixed-length string depends on where the string is declared, as described in the following table:

### Strings Declared Are Stored

|                |                                                                                                                                                                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| In structures  | In the same data area as that of the structure. Local structures are on the stack; public structures are stored in the public data space; and private structures are stored in the private data space. Local structures should be used sparingly as stack space is limited. |
| In arrays      | In the global string space along with all the other array elements.                                                                                                                                                                                                         |
| Local routines | On the stack. The stack is limited in size, so local fixed-length strings should be used sparingly.                                                                                                                                                                         |

**See Also** Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); Variant (data type); Boolean (data type); *DefType* (statement); *CStr* (function).

**Platform(s)** Windows and Macintosh.

## String, String\$ (functions)

**Syntax** String[\$](number, [CharCode | text\$])

**Description** Returns a string of length *number* consisting of a repetition of the specified filler character.

**Comments** String\$ returns a String, whereas String returns a String variant.  
These functions take the following parameters:

| Parameter       | Description                                                                                                                                                                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>number</i>   | Integer specifying the number of repetitions.                                                                                                                                                                                                              |
| <i>CharCode</i> | Integer specifying the character code to be used as the filler character. If <i>CharCode</i> is greater than 255 (the largest character value), then WM Basic converts it to a valid character using the following formula:<br>$\text{CharCode} \bmod 256$ |
| <i>text\$</i>   | Any String expression, the first character of which is used as the filler character.                                                                                                                                                                       |

**Example** 'This example uses the String function to create a line of "=" signs 'the length of another string and then displays the character string 'underlined with the generated string.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 a$ = "This string will appear underlined."
 b$ = String$(Len(a$), "=")
 MsgBox a$ & crlf & b$
End Sub
```

**See Also** Space, Space\$ (functions).

**Platform(s)** Windows and Macintosh.

---

## Sub...End Sub (statement)

---

**Syntax** [Private | Public] [Static] Sub *name*[(*arglist*)]  
    [*statements*]  
End Sub

Where *arglist* is a comma-separated list of the following (up to 30 arguments are allowed):

[Optional] [ByVal | ByRef] *parameter*[()] [As *type*]

**Description** Declares a subroutine.

**Comments** The Sub statement has the following parts:

| Part                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Private</code>  | Indicates that the subroutine being defined cannot be called from other scripts.                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>Public</code>   | Indicates that the subroutine being defined can be called from other scripts. If the <code>Private</code> and <code>Public</code> keywords are both missing, then <code>Public</code> is assumed.                                                                                                                                                                                                                                                                                 |
| <code>Static</code>   | Recognized by the compiler but currently has no effect.                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>name</i>           | <p>Name of the subroutine, which must follow WM Basic naming conventions:</p> <ol style="list-style-type: none"> <li>1. Must start with a letter.</li> <li>2. May contain letters, digits, and the underscore character (<code>_</code>). Punctuation and type-declaration characters are not allowed. The exclamation point (<code>!</code>) can appear within the name as long as it is not the last character.</li> <li>3. Must not exceed 80 characters in length.</li> </ol> |
| <code>Optional</code> | <p>Keyword indicating that the parameter is optional. All optional parameters must be of type <code>Variant</code>. Furthermore, all parameters that follow the first optional parameter must also be optional.</p> <p>If this keyword is omitted, then the parameter is required.</p> <hr/> <p><b>Note:</b> You can use the <code>IsMissing</code> function to determine if an optional parameter was actually passed by the caller.</p> <hr/>                                   |
| <code>ByVal</code>    | Keyword indicating that the parameter is passed by value.                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <code>ByRef</code>    | Keyword indicating that the parameter is passed by reference. If neither the <code>ByVal</code> nor the <code>ByRef</code> keyword is given, then <code>ByRef</code> is assumed.                                                                                                                                                                                                                                                                                                  |
| <i>parameter</i>      | Name of the parameter, which must follow the same naming conventions as those used by variables. This name can include a type-declaration character, appearing in place of <code>As type</code> .                                                                                                                                                                                                                                                                                 |
| <i>type</i>           | <p>Type of the parameter (i.e., <code>Integer</code>, <code>String</code>, and so on). Arrays are indicated with parentheses. For example, an array of integers would be declared as follows:</p> <pre>Sub Test(a() As Integer) End Sub</pre> <p>A subroutine terminates when one of the following statements is encountered:</p> <pre>End Sub Exit Sub</pre> <p>Subroutines can be recursive.</p>                                                                                |

### Passing Parameters to Subroutines

Parameters are passed to a subroutine either by value or by reference, depending on the declaration of that parameter in *arglist*. If the parameter is declared using the `ByRef` keyword, then any modifications to that passed parameter within the subroutine change the value of that variable in the caller. If the parameter is declared using the `ByVal` keyword, then the value of that variable cannot be changed in the called subroutine. If neither the `ByRef` or `ByVal` keywords are specified, then the parameter is passed by reference.

You can override passing a parameter by reference by enclosing that parameter within parentheses. For instance, the following example passes the variable `j` by reference, regardless of how the third parameter is declared in the *arglist* of `UserSub`:

```
UserSub 10,12,(j)
```

### Optional Parameters

WM Basic allows you to skip parameters when calling subroutines, as shown in the following example:

```
Sub Test(a%,b%,c%)
End Sub

Sub Main
 Test 1,,4 'Parameter 2 was skipped.
End Sub
```

You can skip any parameter with the following restrictions:

1. The call cannot end with a comma. For instance, using the above example, the following is not valid:

```
Test 1,,
```

2. The call must contain the minimum number of parameters as required by the called subroutine. For instance, using the above example, the following are invalid:

```
Test ,1 'Only passes two out of three required
parameters.
Test 1,2 'Only passes two out of three required parameters.
```



When you skip a parameter in this manner, WM Basic creates a temporary variable and passes this variable instead. The value of this temporary variable depends on the data type of the corresponding parameter in the argument list of the called subroutine, as described in the following table:

| Value              | Data type                               |
|--------------------|-----------------------------------------|
| 0                  | Integer, Long, Single, Double, Currency |
| Zero-length string | String                                  |
| Nothing            | Object (or any data object)             |
| Error              | Variant                                 |
| December 30, 1899  | Date                                    |
| False              | Boolean                                 |

Within the called subroutine, you will be unable to determine if a parameter was skipped unless the parameter was declared as a variant in the argument list of the subroutine. In this case, you can use the `IsMissing` function to determine if the parameter was skipped:

```
Sub Test(a,b,c)
 If IsMissing(a) Or IsMissing(b) Then Exit Sub
End Sub
```

**Example** 'This example uses a subroutine to calculate the area of a circle.

```
Sub Main()
 r! = 10
 PrintArea r!
End Sub

Sub PrintArea(r as single)
 area! = (r! ^ 2) * Pi
 MsgBox "The area of a circle with radius " & r! & " = " & area!
End Sub
```

**See Also** `Main` (keyword); `Function...End Function` (statement).

**Platform(s)** Windows and Macintosh.

## Switch (function)

**Syntax** `Switch(condition1,expression1 [ ,condition2,expression2 ... [ ,condition7,expression7]])`

**Description** Returns the expression corresponding to the first `True` condition.

**Comments** The `Switch` function evaluates each condition and expression, returning the expression that corresponds to the first condition (starting from the left) that evaluates to `True`. Up to seven condition/expression pairs can be specified.

A runtime error is generated if there is an odd number of parameters (i.e., there is a condition without a corresponding expression).

The `Switch` function returns `Null` if no condition evaluates to `True`.

**Example** 'The following code fragment displays the current operating platform.  
If the  
'platform is unknown, then the word "Unknown" is displayed.

```
Sub Main()
 Dim a As Variant
 a = Switch(Basic.OS = 0,"Windows 3.1",Basic.OS = 10,"Mac")
 MsgBox "The current platforms is: " & IIf(IsNull(a),"Unknown",a)
End Sub
```

**See Also** `Choose (function)`; `IIf (function)`; `If...Then...Else (statement)`;  
`Select...Case (statement)`.

**Platform(s)** Windows and Macintosh.

## SYD (function)

---

**Syntax** `SYD ( Cost , Salvage , Life , Period )`

**Description** Returns the sum of years' digits depreciation of an asset over a specific period of time.

**Comments** The SYD of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers.

The formula used to find the SYD of an asset is as follows:

$$(\text{Cost} - \text{Salvage\_Value}) * \text{Remaining\_Useful\_Life} / \text{SYD}$$

The SYD function requires the following parameters:

| Parameter                                                                                                                                                                                                                                 | Description                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <i>Cost</i>                                                                                                                                                                                                                               | Double representing the initial cost of the asset.                                                                     |
| <i>Salvage</i>                                                                                                                                                                                                                            | Double representing the estimated value of the asset at the end of its useful life.                                    |
| <i>Life</i>                                                                                                                                                                                                                               | Double representing the length of the asset's useful life.                                                             |
| <i>Period</i>                                                                                                                                                                                                                             | Double representing the period for which the depreciation is to be calculated. It cannot exceed the life of the asset. |
| To receive accurate results, the parameters <i>Life</i> and <i>Period</i> must be expressed in the same units. If <i>Life</i> is expressed in terms of months, for example, then <i>Period</i> must also be expressed in terms of months. |                                                                                                                        |

**Example** 'In this example, an asset that cost \$1,000.00 is depreciated over ten years.  
'The salvage value is \$100.00, and the sum of the years' digits depreciation is shown for each year.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 For x = 1 To 10
 dep# = SYD(1000,100,10,x)
 msg = msg & "Year: " & x & " Dep: " & Format(dep#,"Currency") & crlf
 Next x
 MsgBox msg
End Sub
```

**See Also** Sln (function); DDB (function).

**Platform(s)** Windows and Macintosh.

## System.Exit (method)

**Syntax** System.Exit

**Description** Exits the operating environment.

**Example** 'This example asks whether the user would like to restart Windows after exiting.

```
Sub Main
 button = MsgBox("Restart Windows on exit?",ebYesNo,"Exit Windows")
 If button = ebYes Then System.Restart 'Yes button selected.
 If button = ebNo Then System.Exit 'No button selected.
End Sub
```

**See Also** System.Restart (method).

**Platform(s)** Windows.

## System.FreeMemory (property)

**Syntax** System.FreeMemory

**Description** Returns a Long indicating the number of bytes of free memory.

**Example** 'The following example gets the free memory and converts it to kilobytes

```
Sub Main()
 FreeMem& = System.FreeMemory
 FreeKBytes$ = Format(FreeMem& / 1000,"###,###")
 MsgBox FreeKBytes$ & " Kbytes of free memory"
End Sub
```

**See Also** System.TotalMemory (property); System.FreeResources (property);  
Basic.FreeMemory (property).

**Platform(s)** Windows.

## System.FreeResources (property)

---

**Syntax** System.FreeResources

**Description** Returns an Integer representing the percentage of free system resources.

**Comments** The returned value is between 0 and 100.

**Example** 'This example gets the percentage of free resources.

```
Sub Main()
 FreeRes% = System.FreeResources
 MsgBox FreeRes% & "% of memory resources available."
End Sub
```

**See Also** System.TotalMemory (property); System.FreeMemory (property);  
Basic.FreeMemory (property).

**Platform(s)** Windows.

## System.MouseTrails (method)

---

**Syntax** System.MouseTrails *isOn*

**Description** Toggles mouse trails on or off.

**Comments** If *isOn* is True, then mouse trails are turned on; otherwise, mouse trails are turned off.

A runtime error is generated if mouse trails is not supported on your system.

**Example** 'This example turns on mouse trails.

```
Sub Main
 System.MouseTrails 1
End Sub
```

**Platform(s)** Windows.

**Platform** Under Windows, the setting is saved in the INI file permanently.

**Notes:**  
Windows

## System.Restart (method)

---

**Syntax** System.Restart

**Description** Restarts the operating environment.

**Example** 'This example asks whether the user would like to restart Windows after exiting.

```
Sub Main
 button = MsgBox ("Restart Windows on exit?", vbYesNo, "Exit Windows")
 If button = vbYes Then System.Restart 'Yes button selected.
 If button = vbNo Then System.Exit 'No button selected.
End Sub
```

**See Also** System.Exit (method).

**Platform(s)** Windows.

## System.TotalMemory (property)

**Syntax** System.TotalMemory

**Description** Returns a Long representing the number of bytes of available free memory in Windows.

**Example** 'This example displays the total system memory.

```
Sub Main()
 TotMem& = System.TotalMemory
 TotKBytes$ = Format(TotMem& / 1000, "###,###")
 MsgBox TotKBytes$ & " Kbytes of total system memory exist"
End Sub
```

**See Also** System.FreeMemory (property); System.FreeResources (property); Basic.FreeMemory (property).

**Platform(s)** Windows.

## System.WindowsDirectory\$ (property)

**Syntax** System.WindowsDirectory\$

**Description** Returns the home directory of the operating environment.

**Example** 'This example displays the windows directory.

```
Sub Main
 MsgBox "Windows directory = " & System.WindowsDirectory$
End Sub
```

**See Also** Basic.HomeDir\$ (property).

**Platform(s)** Windows.

## System.WindowsVersion\$ (property)

**Syntax** System.WindowsVersion\$

**Description** Returns the version of the operating environment, such as "3.0" or "3.1."

**Example**    'This example sets the UseWin31 variable to True if the Windows version is  
              'greater than or equal to 3.1; otherwise, it sets the UseWin31 variable  
              'to False.

```
Sub Main()
 If Val(System.WindowsVersion$) > 3.1 Then
 MsgBox "You are running a Windows version later than 3.1"
 Else
 MsgBox "You are running Windows version 3.1 or earlier"
 End If
End Sub
```

**See Also**    Basic.Version\$ (property).

**Platform(s)**    Windows.

---

## Tab (function)

---

**Syntax** `Tab(column)`

**Description** Prints the number of spaces necessary to reach a given column position.

**Comments** This function can only be used with the `Print` and `Print#` statements.

The *column* parameter is an `Integer` specifying the desired column position to which to advance. It can be any value between 0 and 32767 inclusive.

**Rule 1:** If the current print position is less than or equal to *column*, then the number of spaces is calculated as:

`column - print_position`

**Rule 2:** If the current print position is greater than *column*, then *column* - 1 spaces are printed on the next line.

If a line width is specified (using the `Width` statement), then the column position is adjusted as follows before applying the above two rules:

`column = column Mod width`

The `Tab` function is useful for making sure that output begins at a given column position, regardless of the length of the data already printed on that line.

**Example** 'This example prints three column headers and three numbers  
'aligned below the column headers.

```
Sub Main()
 ViewportOpen
 Print "Column1";Tab(10);"Column2";Tab(20);"Column3"
 Print Tab(3);"1";Tab(14);"2";Tab(24);"3"
 Sleep(10000) 'Wait 10 seconds.
 ViewportClose
End Sub
```

**See Also** `Spc (function)`; `Print (statement)`; `Print# (statement)`.

**Platform(s)** Windows and Macintosh.

---

## Tan (function)

---

**Syntax** `Tan(angle)`

**Description** Returns a `Double` representing the tangent of *angle*.

**Comments** The *angle* parameter is a `Double` value given in radians.

**Example** 'This example computes the tangent of pi/4 radians (45 degrees).

```
Sub Main()
 c# = Tan(Pi / 4)
 MsgBox "The tangent of 45 degrees is: " & c#
End Sub
```

**See Also** Sin (function); Cos (function); Atn (function).

**Platform(s)** Windows and Macintosh.

---

## Text (statement)

---

**Syntax** Text *x,y,width,height,title\$* [ , [ *.Identifier* ] [ , [ *FontName\$* ] [ , [ *size* ] [ , *style* ] ] ] ]

**Description** Defines a text control within a dialog box template. The text control only displays text; the user cannot set the focus to a text control or otherwise interact with it.

**Comments** The text within a text control word-wraps. Text controls can be used to display up to 32K of text.

The Text statement accepts the following parameters:

| Parameter            | Description                                                                                                                                                                                                                                                                                      |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>x, y</i>          | Integer positions of the control (in dialog units) relative to the upper left corner of the dialog box.                                                                                                                                                                                          |
| <i>width, height</i> | Integer dimensions of the control in dialog units.                                                                                                                                                                                                                                               |
| <i>title\$</i>       | String containing the text that appears within the text control. This text may contain an ampersand character to denote an accelerator letter, such as "&Save" for Save. Pressing this accelerator letter sets the focus to the control following the Text statement in the dialog box template. |
| <i>Identifier</i>    | Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the first two words from <i>title\$</i> are used.                                                                                                             |
| <i>FontName\$</i>    | Name of the font used for display of the text within the text control. If omitted, then the default font for the dialog is used.                                                                                                                                                                 |



*size* Size of the font used for display of the text within the text control. If omitted, then the default size for the default font of the dialog is used.

*style* Style of the font used for display of the text within the text control. This can be any of the following values:

|              |                                             |
|--------------|---------------------------------------------|
| ebRegular    | Normal font (i.e., neither bold nor italic) |
| ebBold       | Bold font                                   |
| ebItalic     | Italic font                                 |
| ebBoldItalic | Bold-italic font                            |

If omitted, then ebRegular is used.

**Example**

```
Begin Dialog UserDialog3 81,64,128,60,"Untitled"
 CancelButton 80,32,40,14
 OKButton 80,8,40,14
 Text 4,8,68,44,"This text is displayed in the dialog box."
End Dialog
```

**See Also** CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); TextBox (statement); Begin Dialog (statement), PictureButton (statement).

**Platform(s)** Windows and Macintosh.

**Platform Notes:** Under Windows, accelerators are underlined, and the Alt+*letter* accelerator combination is used.

**Windows** Under Windows, 8-point MS Sans Serif is the default font used within user dialogs.

**Platform Notes:** On the Macintosh, accelerators are normal in appearance, and the Command+*letter* accelerator combination is used.

**Macintosh** On the Macintosh, 10-point Geneva is the default font used within user dialogs.

## TextBox (statement)

**Syntax** `TextBox x,y,width,height, .Identifier [ , [isMultiline] [ , [FontName$] [ , [size] [ , style] ] ] ]`

**Description** Defines a single or multiline text-entry field within a dialog box template.

**Comments** If *isMultiline* is 1, the `TextBox` statement creates a multiline text-entry field. When the user types into a multiline field, pressing the Enter key creates a new line rather than selecting the default button.

This statement can only appear within a dialog box template (i.e., between the `Begin Dialog` and `End Dialog` statements).

The `TextBox` statement requires the following parameters:

| Parameter                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                        |                                             |                     |           |                       |             |                           |                  |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|---------------------------------------------|---------------------|-----------|-----------------------|-------------|---------------------------|------------------|
| <i>x, y</i>               | Integer position of the control (in dialog units) relative to the upper left corner of the dialog box.                                                                                                                                                                                                                                                                                                                                                           |                        |                                             |                     |           |                       |             |                           |                  |
| <i>width, height</i>      | Integer dimensions of the control in dialog units.                                                                                                                                                                                                                                                                                                                                                                                                               |                        |                                             |                     |           |                       |             |                           |                  |
| <i>Identifier</i>         | Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code> ). This parameter also creates a string variable whose value corresponds to the content of the text box. This variable can be accessed using the syntax:<br><div style="text-align: center;"><i>DialogVariable . Identifier</i></div>                                                                                  |                        |                                             |                     |           |                       |             |                           |                  |
| <i>isMultiline</i>        | Specifies whether the text box can contain more than a single line (0 = single-line; 1 = multiline).                                                                                                                                                                                                                                                                                                                                                             |                        |                                             |                     |           |                       |             |                           |                  |
| <i>FontName\$</i>         | Name of the font used for display of the text within the text box control. If omitted, then the default font for the dialog is used.                                                                                                                                                                                                                                                                                                                             |                        |                                             |                     |           |                       |             |                           |                  |
| <i>size</i>               | Size of the font used for display of the text within the text box control. If omitted, then the default size for the default font of the dialog is used.                                                                                                                                                                                                                                                                                                         |                        |                                             |                     |           |                       |             |                           |                  |
| <i>style</i>              | Style of the font used for display of the text within the text box control. This can be any of the following values:<br><div style="text-align: center;"><table><tr><td><code>ebRegular</code></td><td>Normal font (i.e., neither bold nor italic)</td></tr><tr><td><code>ebBold</code></td><td>Bold font</td></tr><tr><td><code>ebItalic</code></td><td>Italic font</td></tr><tr><td><code>ebBoldItalic</code></td><td>Bold-italic font</td></tr></table></div> | <code>ebRegular</code> | Normal font (i.e., neither bold nor italic) | <code>ebBold</code> | Bold font | <code>ebItalic</code> | Italic font | <code>ebBoldItalic</code> | Bold-italic font |
| <code>ebRegular</code>    | Normal font (i.e., neither bold nor italic)                                                                                                                                                                                                                                                                                                                                                                                                                      |                        |                                             |                     |           |                       |             |                           |                  |
| <code>ebBold</code>       | Bold font                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                        |                                             |                     |           |                       |             |                           |                  |
| <code>ebItalic</code>     | Italic font                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                        |                                             |                     |           |                       |             |                           |                  |
| <code>ebBoldItalic</code> | Bold-italic font                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                        |                                             |                     |           |                       |             |                           |                  |

If omitted, then `ebRegular` is used.

When the dialog box is created, the *Identifier* variable is used to set the initial content of the text box. When the dialog box is dismissed, the variable will contain the new content of the text box.

A single-line text box can contain up to 256 characters. The length of text in a multiline text box is not limited by WM Basic; the default memory limit specified by the given platform is used instead.

**Example** `Begin Dialog UserDialog3 81,64,128,60,"Untitled"  
 CancelButton 80,32,40,14  
 OKButton 80,8,40,14  
 TextBox 4,8,68,44,.TextBox1,1  
End Dialog`

**See Also** `CancelButton (statement); CheckBox (statement); ComboBox (statement); Dialog (function); Dialog (statement); DropListBox (statement); GroupBox (statement); ListBox (statement); OKButton (statement); OptionButton (statement); OptionGroup (statement); Picture (statement); PushButton (statement); Text (statement); Begin Dialog (statement), PictureButton (statement).`

**Platform(s)** Windows and Macintosh.

**Platform Notes:** Under Windows, 8-point MS Sans Serif is the default font used within user dialogs.

**Windows**

**Platform Notes:** On the Macintosh, 10-point Geneva is the default font used within user dialogs.

**Macintosh**

## Time, Time\$ (functions)

**Syntax** `Time[$] [()]`

**Description** Returns the system time as a String or as a Date variant.

**Comments** The Time\$ function returns a String contains the time in 24-hour time format, whereas Time returns a Date variant.

To set the time, use the Time/Time\$ statements.

**Example** 'This example returns the system time and displays it in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 oldtime$ = Time$
 msg = "Time was: " & oldtime$ & crlf
 Time$ = "10:30:54"
 msg = msg & "Time set to: " & Time$ & crlf
 Time$ = oldtime$
 msg = msg & "Time restored to: " & Time$
 MsgBox msg
End Sub
```

**See Also** `Time, Time$ (statements); Date, Date$ (functions); Date, Date$ (statements); Now (function).`

**Platform(s)** Windows and Macintosh.

---

## Time, Time\$ (statements)

---

**Syntax** Time[\$] = *newtime*

**Description** Sets the system time to the time contained in the specified string.

**Comments** The Time\$ statement requires a string variable in one of the following formats:

*HH*  
*HH:MM*  
*HH:MM:SS*

where *HH* is between 0 and 23, *MM* is between 0 and 59, and *SS* is between 0 and 59.

The Time statement converts any valid expression to a time, including string and numeric values. Unlike the Time\$ statement, Time recognizes many different time formats, including 12-hour times.

**Example** 'This example returns the system time and displays it in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 oldtime$ = Time$
 msg = "Time was: " & oldtime$ & crlf
 Time$ = "10:30:54"
 msg = msg & "Time set to: " & Time$ & crlf
 Time$ = oldtime$
 msg = msg & "Time restored to: " & Time$
 MsgBox msg
End Sub
```

**See Also** Time, Time\$ (functions); Date, Date\$ (functions); Date, Date\$ (statements).

**Platform(s)** Windows and Macintosh.

---

## Timer (function)

---

**Syntax** Timer

**Description** Returns a Single representing the number of seconds that have elapsed since midnight.

**Example** 'This example displays the elapsed time between execution start and 'the time you clicked the OK button on the first message.

```
Sub Main()
 start& = Timer
 MsgBox "Click the OK button, please."
 total& = Timer - start&
 MsgBox "The elapsed time was: " & total& & " seconds."
End Sub
```

**See Also** Time, Time\$ (functions); Now (function).

**Platform(s)** Windows and Macintosh.

## TimeSerial (function)

**Syntax** TimeSerial(*hour,minute,second*)

**Description** Returns a Date variant representing the given time with a date of zero.

**Comments** The TimeSerial function requires the following parameters:

| Parameter | Description |
|-----------|-------------|
|-----------|-------------|

|             |                           |
|-------------|---------------------------|
| <i>hour</i> | Integer between 0 and 23. |
|-------------|---------------------------|

|               |                           |
|---------------|---------------------------|
| <i>minute</i> | Integer between 0 and 59. |
|---------------|---------------------------|

|               |                           |
|---------------|---------------------------|
| <i>second</i> | Integer between 0 and 59. |
|---------------|---------------------------|

**Example**

```
Sub Main()
 start# = TimeSerial(10,22,30)
 finish# = TimeSerial(10,35,27)
 dif# = Abs(start# - finish#)
 MsgBox "The time difference is: " & Format(dif#, "hh:mm:ss")
End Sub
```

**See Also** DateValue (function); TimeValue (function); DateSerial (function).

**Platform(s)** Windows and Macintosh.

## TimeValue (function)

**Syntax** TimeValue(*time\_string\$*)

**Description** Returns a Date variant representing the time contained in the specified string argument.

**Comments** This function interprets the passed *time\_string\$* parameter looking for a valid time specification.

The *time\_string\$* parameter can contain valid time items separated by time separators such as colon (:) or period (.).

Time strings can contain an optional date specification, but this is not used in the formation of the returned value.

If a particular time item is missing, then it is set to 0. For example, the string "10 pm" would be interpreted as "22:00:00."

**Example** 'This example calculates the TimeValue of the current time and  
'displays it in a dialog box.

```
Sub Main()
 t1$ = "10:15"
 t2# = TimeValue(t1$)
 MsgBox "The TimeValue of " & t1$ & " is: " & t2#
End Sub
```

**See Also** DateValue (function); TimeSerial (function); DateSerial (function).

**Platform(s)** Windows and Macintosh.

**Platform Notes:** Under Windows, time specifications vary, depending on the international settings contained in the [ intl ] section of the win.ini file.

**Windows**

## Trim, Trim\$ (functions)

---

**Syntax** Trim[\$](*text*)

**Description** Returns a copy of the passed string expression (*text*) with leading and trailing spaces removed.

**Comments** Trim\$ returns a String, whereas Trim returns a String variant.

Null is returned if *text* is Null.

**Example** 'This example uses the Trim\$ function to extract the nonblank part  
'of a string and display it.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 text$ = " This is text "
 tr$ = Trim$(text$)
 MsgBox "Original =>" & text$ & "<=" & crlf & "Trimmed =>" & tr$ &
 "<=" &
End Sub
```

**See Also** LTrim, LTrim\$ (functions); RTrim, RTrim\$ (functions).

**Platform(s)** Windows and Macintosh.

## True (constant)

---

**Description** Boolean constant whose value is True.

**Comments** Used in conditionals and Boolean expressions.

**Example** 'This example sets variable a to True and then tests to see whether  
'(1) A is True; (2) the True constant = -1; and (3) A is  
'equal to -1 (True).

```
Sub Main()
 a = True
 If ((a = True) and (True = -1) and (a = -1)) then
 MsgBox "a is True."
 Else
 MsgBox "a is False."
 End If
End Sub
```

**See Also** False (constant); Constants (topic); Boolean (data type).

**Platform(s)** Windows and Macintosh.

## Type (statement)

**Syntax** *Type username*  
    *variable As type*  
    *variable As type*  
    *variable As type*  
    :  
End *Type*

**Description** The Type statement creates a structure definition that can then be used with the Dim statement to declare variables of that type. The *username* field specifies the name of the structure that is used later with the Dim statement.

**Comments** Within a structure definition appear field descriptions in the format:

*variable As type*

where *variable* is the name of a field of the structure, and *type* is the data type for that variable. Any fundamental data type or previously declared user-defined data type can be used within the structure definition (structures within structures are allowed). Only fixed arrays can appear within structure definitions.

The Type statement can only appear outside of subroutine and function declarations.

When declaring strings within fixed-size types, it is useful to declare the strings as fixed-length. Fixed-length strings are stored within the structure itself rather than in the string space. For example, the following structure will always require 62 bytes of storage:

```
Type Person
 FirstName As String * 20
 LastName As String * 40
 Age As Integer
End Type
```

---

**Note:** Fixed-length strings within structures are size-adjusted upward to an even byte boundary. Thus, a fixed-length string of length 5 will occupy 6 bytes of storage within the structure.

---

**Example** 'This example displays the use of the Type statement to create a 'structure representing the parts of a circle and assign values 'to them.

```
Type Circ
 msg As String
 rad As Integer
 dia As Integer
 are As Double
 cir As Double
End Type

Sub Main()
 Dim circle As Circ
 circle.rad = 5
 circle.dia = circle.rad * 2
 circle.are = (circle.rad ^ 2) * Pi
 circle.cir = circle.dia * Pi
 circle.msg = "The area of the circle is: " & circle.are
 MsgBox circle.msg
End Sub
```

**See Also** Dim (statement); Public (statement); Private (statement).

**Platform(s)** Windows and Macintosh.

---

## UBound (function)

---

**Syntax** UBound(ArrayVariable ( ) [ , dimension ])

**Description** Returns an Integer containing the upper bound of the specified dimension of the specified array variable.



**Comments** The *dimension* parameter is an integer that specifies the desired dimension. If not specified, then the upper bound of the first dimension is returned.

The `UBound` function can be used to find the upper bound of a dimension of an array returned by an OLE automation method or property:

```
UBound(object.property [,dimension])
```

```
UBound(object.method [,dimension])
```

**Example** 'This example dimensions two arrays and displays their upper bounds.

```
Sub Main()
 Dim a(5 To 12)
 Dim b(2 To 100, 9 To 20)
 uba = UBound(a)
 ubb = UBound(b,2)
 MsgBox "The upper bound of a is: " & uba & " The upper bound of b
is: " & ubb
```

'This example uses `Lbound` and `Ubound` to dimension a dynamic array to hold a copy of an array redimmed by the `FileList` statement.

```
Dim fl$()
FileList fl$,"*"
count = Ubound(fl$)
If ArrayDims(a) Then
 Redim nl$(Lbound(fl$) To Ubound(fl$))
 For x = 1 To count
 nl$(x) = fl$(x)
 Next x
 MsgBox "The last element of the new array is: " & nl$(count)
End If
End Sub
```

**See Also** `LBound` (function); `ArrayDims` (function); Arrays (topic).

**Platform(s)** Windows and Macintosh.

## UCase, UCase\$ (functions)

**Syntax** `UCase[$](text)`

**Description** Returns the uppercase equivalent of the specified string.

**Comments** `UCase$` returns a `String`, whereas `UCase` returns a `String` variant.

`Null` is returned if *text* is `Null`.

**Example** 'This example uses the UCase\$ function to change a string from lowercase to uppercase.

```
Sub Main()
 a1$ = "this string was lowercase, but was converted."
 a2$ = UCase$(a1$)
 MsgBox a2$
End Sub
```

**See Also** LCase, LCase\$ (functions).

**Platform(s)** Windows and Macintosh.

## Unlock (statement)

---

**Syntax** Unlock [#] *filename* [, {*record* | [*start*] To *end*}]

**Description** Unlocks a section of the specified file, allowing other processes access to that section of the file.

**Comments** The Unlock statement requires the following parameters:

| Parameter       | Description                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>filename</i> | Integer used by WM Basic to refer to the open file—the number passed to the Open statement.                                  |
| <i>record</i>   | Long specifying which record to unlock.                                                                                      |
| <i>start</i>    | Long specifying the first record within a range to be unlocked.                                                              |
| <i>end</i>      | Long specifying the last record within a range to be unlocked.                                                               |
|                 | For sequential files, the <i>record</i> , <i>start</i> , and <i>end</i> parameters are ignored: the entire file is unlocked. |

The section of the file is specified using one of the following:

| Syntax                  | Description                                                                                                  |
|-------------------------|--------------------------------------------------------------------------------------------------------------|
| No record specification | Unlock the entire file.                                                                                      |
| <i>record</i>           | Unlock the specified record number (for Random files) or byte (for Binary files).                            |
| <i>to end</i>           | Unlock from the beginning of the file to the specified record (for Random files) or byte (for Binary files). |
| <i>start to end</i>     | Unlock the specified range of records (for Random files) or bytes (for Binary files).                        |

The unlock range must be the same as that used by the Lock statement.

**Example** 'This example creates a file named test.dat and fills it with ten 'string variable records. These are displayed in a dialog box. The 'file is then reopened for read/write, and each record is locked, 'modified, rewritten, and unlocked. The new records are then 'displayed in a dialog box.

```
Const crlf = Chr$(13) + Chr$(10)

Sub Main()
 a$ = "This is record number: "
 b$ = "0"
 rec$ = ""

 msg = ""
 Open "test.dat" For Random Access Write Shared As #1
 For x = 1 To 10
 rec$ = a$ & x
 Lock #1,x
 Put #1,,rec$
 UnLock #1,x
 msg = msg & rec$ & crlf
 Next x
 Close
 MsgBox "The records are: " & crlf & msg

 msg = ""
 Open "test.dat" For Random Access Read Write Shared As #1
 For x = 1 to 10
 rec$ = Mid$(rec$,1,23) & (11 - x)
 Lock #1,x 'Lock it for our use.
 Put #1,x,rec$ 'Nobody's changed it.
 UnLock #1,x
 msg = msg & rec$ & crlf
 Next x
 MsgBox "The records are: " & crlf & msg
 Close

 Kill "test.dat"
End Sub
```

**See Also** Lock (statement); Open (statement).

**Platform(s)** Windows and Macintosh.

**Platform** On the Macintosh, file locking will only succeed on volumes that are shared  
**Notes:** (i.e., file sharing is on).

**Macintosh**

## User-Defined Types (topic)

*User-defined types* (UDTs) are structure definitions created using the `Type` statement. UDTs are equivalent to C language structures.

### Declaring Structures

The `Type` statement is used to create a structure definition. Type declarations must appear outside the body of all subroutines and functions within a script and are therefore global to an entire script.

Once defined, a UDT can be used to declare variables of that type using the `Dim`, `Public`, or `Private` statement. The following example defines a rectangle structure:

```
Type Rect
 left As Integer
 top As Integer
 right As Integer
 bottom As Integer
End Type
:
Sub Main()
 Dim r As Rect
 :
 r.left = 10
End Sub
```

Any fundamental data type can be used as a structure member, including other user-defined types. Only fixed arrays can be used within structures.

### Copying Structures

UDTs of the same type can be assigned to each other, copying the contents. No other standard operators can be applied to UDTs.

```
Dim r1 As Rect
Dim r2 As Rect
:
r1 = r2
```

When copying structures of the same type, all strings in the source UDT are duplicated and references are placed into the target UDT.

The `LSet` statement can be used to copy a UDT variable of one type to another:

```
LSet variable1 = variable2
```

`LSet` cannot be used with UDTs containing variable-length strings. The smaller of the two structures determines how many bytes get copied.

### Passing Structures

UDTs can be passed both to user-defined routines and to external routines, and they can be assigned. UDTs are always passed by reference.

Since structures are always passed by reference, the `ByVal` keyword cannot be used when defining structure arguments passed to external routines (using `Declare`). The `ByVal` keyword can only be used with fundamental data types such as `Integer` and `String`.

Passing structures to external routines actually passes a far pointer to the data structure.

### Size of Structures

The `Len` function can be used to determine the number of bytes occupied by a UDT:

```
Len(udt_variable_name)
```

Since strings are stored in WM Basic's data space, only a reference (currently, 2 bytes) is stored within a structure. Thus, the `Len` function may seem to return incorrect information for structures containing strings.

---

## Val (function)

**Syntax** `Val(number)`

**Description** Converts a given string expression to a number.

**Comments** The *number* parameter can contain any of the following:

- Leading minus sign (for nonhex or octal numbers only)
- Hexadecimal number in the format *&Hhexdigits*
- Octal number in the format *&Ooctaldigits*
- Floating-point number, which can contain a decimal point and an optional exponent

Spaces, tabs, and line feeds are ignored.

If *number* does not contain a number, then 0 is returned.

The `Val` function continues to read characters from the string up to the first nonnumeric character.

The `Val` function always returns a double-precision floating-point value. This value is forced to the data type of the assigned variable.

**Example** 'This example inputs a number string from an InputBox and converts it to a number variable.

```
Sub Main()
 a$ = InputBox$("Enter anything containing a number","Enter Number")
 b# = Val(a$)
 MsgBox "The value is: " & b#
End Sub

'The following table shows valid strings and their numeric equivalents:
' "1 2 3" 123
' "12.3" 12.3
' "&HFFFF" -1
' "&O77" 63
' "12.345E-02" .12345
```

**See Also** CDBl (function); Str, Str\$ (functions).

**Platform(s)** Windows and Macintosh.

## Variant (data type)

**Syntax** Variant

**Description** A data type used to declare variables that can hold one of many different types of data.

**Comments** During a variant's existence, the type of data contained within it can change. Variants can contain any of the following types of data:

| Type of Data    | WM Basic Data Types                                     |
|-----------------|---------------------------------------------------------|
| Numeric         | Integer, Long, Single, Double, Boolean, Date, Currency. |
| Logical         | Boolean.                                                |
| Dates and times | Date.                                                   |
| String          | String.                                                 |
| Object          | Object.                                                 |
| No valid data   | A variant with no valid data is considered Null.        |
| Uninitialized   | An uninitialized variant is considered Empty.           |

There is no type-declaration character for variants.

The number of significant digits representable by a variant depends on the type of data contained within the variant.

Variant is the default data type for WM Basic. If a variable is not explicitly declared with Dim, Public, or Private, and there is no type-declaration character (i.e., #, @, !, %, or &), then the variable is assumed to be Variant.

**Determining the Subtype of a Variant**

The following functions are used to query the type of data contained within a variant:

| Function  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VarType   | Returns a number representing the type of data contained within the variant.                                                                                                                                                                                                                                                                                                                                                                |
| IsNumeric | Returns <code>True</code> if a variant contains numeric data. The following are considered numeric:<br><br>Integer, Long, Single, Double, Date, Boolean, Currency<br><br>If a variant contains a string, this function returns <code>True</code> if the string can be converted to a number.<br><br>If a variant contains an <code>Object</code> whose default property is numeric, then <code>IsNumeric</code> returns <code>True</code> . |
| IsObject  | Returns <code>True</code> if a variant contains an object.                                                                                                                                                                                                                                                                                                                                                                                  |
| IsNull    | Returns <code>True</code> if a variant contains no valid data.                                                                                                                                                                                                                                                                                                                                                                              |
| IsEmpty   | Returns <code>True</code> if a variant is uninitialized.                                                                                                                                                                                                                                                                                                                                                                                    |
| IsDate    | Returns <code>True</code> if a variant contains a date. If the variant contains a string, then this function returns <code>True</code> if the string can be converted to a date. If the variant contains an <code>Object</code> , then this function returns <code>True</code> if the default property of that object can be converted to a date.                                                                                           |

**Assigning to Variants**

Before a `Variant` has been assigned a value, it is considered empty. Thus, immediately after declaration, the `VarType` function will return `vbEmpty`. An uninitialized variant is 0 when used in numeric expressions and is a zero-length string when used within string expressions.

A `Variant` is `Empty` only after declaration and before assigning it a value. The only way for a `Variant` to become `Empty` after having received a value is for that variant to be assigned to another `Variant` containing `Empty`, for it to be assigned explicitly to the constant `Empty`, or for it to be erased using the `Erase` statement.

When a variant is assigned a value, it is also assigned that value's type. Thus, in all subsequent operations involving that variant, the variant will behave like the type of data it contains.



### Operations on Variants

Normally, a `Variant` behaves just like the data it contains. One exception to this rule is that, in arithmetic operations, variants are automatically promoted when an overflow occurs. Consider the following statements:

```
Dim a As Integer,b As Integer,c As Integer
Dim x As Variant,y As Variant,z As Variant

a% = 32767
b% = 1
c% = a% + b% 'This will overflow.

x = 32767
y = 1
z = x + y 'z becomes a Long because of Integer overflow.
```

In the above example, the addition involving `Integer` variables overflows because the result (32768) overflows the legal range for integers. With `Variant` variables, on the other hand, the addition operator recognizes the overflow and automatically promotes the result to a `Long`.

### Adding Variants

The `+` operator is defined as performing two functions: when passed strings, it concatenates them; when passed numbers, it adds the numbers.

With variants, the rules are complicated because the types of the variants are not known until execution time. If you use `+`, you may unintentionally perform the wrong operation.

It is recommended that you use the `&` operator if you intend to concatenate two `String` variants. This guarantees that string concatenation will be performed and not addition.

### Variants That Contain No Data

A `Variant` can be set to a special value indicating that it contains no valid data by assigning the `Variant` to `Null`:

```
Dim a As Variant
a = Null
```

The only way that a `Variant` becomes `Null` is if you assign it as shown above.

The `Null` value can be useful for catching errors since its value propagates through an expression.

### Variant Storage

Variants require 16 bytes of storage internally:

- A 2-byte type
- A 2-byte extended type for data objects
- 4 bytes of padding for alignment
- An 8-byte value

Unlike other data types, writing variants to `Binary` or `Random` files does not write 16 bytes. With variants, a 2-byte type is written, followed by the data (2 bytes for `Integer` and so on).

### Disadvantages of Variants

The following list describes some disadvantages of variants:

1. Using variants is slower than using the other fundamental data types (i.e., `Integer`, `Long`, `Single`, `Double`, `Date`, `Object`, `String`, `Currency`, and `Boolean`). Each operation involving a `Variant` requires examination of the variant's type.
2. Variants require more storage than other data types (16 bytes as opposed to 8 bytes for a `Double`, 2 bytes for an `Integer`, and so on).
3. Unpredictable behavior. You may write code to expect an `Integer` variant. At runtime, the variant may be automatically promoted to a `Long` variant, causing your code to break.

### Passing Nonvariant Data to Routines Taking Variants

Passing nonvariant data to a routine that is declared to receive a variant by reference prevents that variant from changing type within that routine. For example:

```
Sub Foo(v As Variant)
 v = 50 'OK.
 v = "Hello, world." 'Get a type-mismatch error here!
End Sub

Sub Main()
 Dim i As Integer
 Foo i 'Pass an integer by reference.
End Sub
```

In the above example, since an `Integer` is passed by reference (meaning that the caller can change the original value of the `Integer`), the caller must ensure that no attempt is made to change the variant's type.

**Passing Variants to Routines Taking Nonvariants**

Variant variables cannot be passed to routines that accept nonvariant data by reference, as demonstrated in the following example:

```
Sub Foo(i as Integer)
End Sub

Sub Main()
 Dim a As Variant
 Foo a 'Compiler gives type-mismatch error here.
End Sub
```

**See Also** Currency (data type); Date (data type); Double (data type); Integer (data type); Long (data type); Object (data type); Single (data type); String (data type); Boolean (data type); DefType (statement); CVar (function); Empty (constant); Null (constant); VarType (function).

**Platform(s)** Windows and Macintosh.

**VarType (function)**

**Syntax** VarType(*variable*)

**Description** Returns an Integer representing the type of data in *variable*.

**Comments** The *variable* parameter is the name of any Variant.

The following table shows the different values that can be returned by VarType:

| Value | Constant     | Data Type                               |
|-------|--------------|-----------------------------------------|
| 0     | ebEmpty      | Uninitialized                           |
| 1     | ebNull       | No valid data                           |
| 2     | ebInteger    | Integer                                 |
| 3     | ebLong       | Long                                    |
| 4     | ebSingle     | Single                                  |
| 5     | ebDouble     | Double                                  |
| 6     | ebCurrency   | Currency                                |
| 7     | ebDate       | Date                                    |
| 8     | ebString     | String                                  |
| 9     | ebObject     | Object (OLE automation object)          |
| 10    | ebError      | User-defined error                      |
| 11    | ebBoolean    | Boolean                                 |
| 12    | ebVariant    | Variant (not returned by this function) |
| 13    | ebDataObject | Non-OLE automation object               |

**Comments** When passed an object, the `VarType` function returns the type of the default property of that object. If the object has no default property, then either `ebObject` or `ebDataObject` is returned, depending on the type of *variable*.

**Example**

```
Sub Main()
 Dim v As Variant
 v = 5& 'Set v to a Long.

 If VarType(v) = ebInteger Then
 MsgBox "v is an Integer."
 ElseIf VarType(v) = ebLong Then
 MsgBox "v is a Long."
 End If
End Sub
```

**See Also** `Empty` (constant); `Null` (constant); `Variant` (data type).

**Platform(s)** Windows and Macintosh.

---

## ViewportClear (statement)

**Syntax** `ViewportClear`

**Description** Clears the open viewport window.

**Comments** The statement has no effect if no viewport is open.

**Example**

```
Sub Main()
 ViewportOpen
 Print "This will be displayed in the viewport window."
 Sleep 2000
 ViewportClear
 Print "This will replace the previous text."
 Sleep 2000
 ViewportClose
End Sub
```

**See Also** `ViewportClose` (statement); `ViewportOpen` (statement).

**Platform(s)** Windows.

---

## ViewportClose (statement)

**Syntax** `ViewportClose`

**Description** This statement closes an open viewport window.

**Comments** The statement has no effect if no viewport is opened.

**Example**

```
Sub Main()
 ViewportOpen
 Print "This will be displayed in the viewport window."
 Sleep 2000
 ViewportClose
End Sub
```

**See Also** ViewportOpen (statement).

**Platform(s)** Windows.

## ViewportOpen (statement)

**Syntax** ViewportOpen [*title\$* [,*x*,*y* [,*width*,*height*]]]

**Description** Opens a new viewport window or switches the focus to the existing viewport window.

**Comments** The ViewportOpen statement requires the following parameters:

| Parameter           | Description                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>title\$</i>      | Specifies a <code>String</code> containing the text to appear in the viewport's caption.                                                                                                                                                                                                                                                                                                        |
| <i>x,y</i>          | Specifies <code>Integer</code> coordinates given in twips indicating the initial position of the upper left corner of the viewport.                                                                                                                                                                                                                                                             |
| <i>width,height</i> | Specifies <code>Integer</code> values indicating the initial width and height of the viewport.<br><br>This statement has no effect if a viewport window is already open.<br><br>Combined with the <code>Print</code> statement, a viewport window is a convenient place to output debugging information.<br><br>The viewport window is closed when the WM Basic host application is terminated. |

The buffer size for the viewport is 32K. Information from the start of the buffer is removed to make room for additional information being appended to the end of the buffer.

The following keys work within a viewport window:

|           |                                                   |
|-----------|---------------------------------------------------|
| Up        | Scrolls up by one line.                           |
| Down      | Scrolls down by one line.                         |
| Home      | Scrolls to the first line in the viewport window. |
| End       | Scrolls to the last line in the viewport window.  |
| PgDn      | Scrolls the viewport window down by one page.     |
| PgUp      | Scrolls the viewport window up by one page.       |
| Ctrl+PgUp | Scrolls the viewport window left by one page.     |
| Ctrl+PgDn | Scrolls the viewport window right by one page.    |

Only one viewport window can be open at any given time. Any scripts with `Print` statements will output information into the same viewport window.

When printing to viewports, the end-of-line character can be any of the following: a carriage return, a line feed, or a carriage-return/line-feed pair.

**Example**

```
Sub Main()
 ViewportOpen "WM Basic Viewport",100,100,500,500
 Print "This will be displayed in the viewport window."
 Sleep 2000
 ViewportClose
End Sub
```

**See Also** `ViewportClose (statement)`.

**Platform(s)** Windows.

---

## VLine (statement)

---

**Syntax** `VLine [lines]`

**Description** Scrolls the window with the focus up or down by the specified number of lines.

**Comments** The *lines* parameter is an `Integer` specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled down by one line.

**Example** 'This example prints a series of lines to the viewport, then 'scrolls back up the lines to the top using VLine.

```
Sub Main()
 ViewportOpen "WM Basic Viewport",100,100,500,200
 For i = 1 to 50
 Print "This will be displayed on line#: " & i
 Next i
 MsgBox "We will now go back 40 lines..."
 VLine -40
 MsgBox "...and here we are!"
 ViewportClose
End Sub
```

**See Also** VPage (statement); VScroll (statement).

**Platform(s)** Windows.

## VPage (statement)

**Syntax** VPage [*pages*]

**Description** Scrolls the window with the focus up or down by the specified number of pages.

**Comments** The *pages* parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled down by one page.

**Example** 'This example scrolls the viewport window up five pages.

```
Sub Main()
 ViewportOpen "WM Basic Viewport",100,100,500,200
 For i = 1 to 500
 Print "This will be displayed on line#: " & i
 Next i
 MsgBox "We will now go back 5 pages..."
 VLine -5
 MsgBox "...and here we are!"
 ViewportClose
End Sub
```

**See Also** VLine (statement); VScroll (statement).

**Platform(s)** Windows.

## VScroll (statement)

**Syntax** VScroll *percentage*

**Description** Sets the thumb mark on the vertical scroll bar attached to the current window.

**Comments** The position is given as a percentage of the total range associated with that scroll bar. For example, if the *percentage* parameter is 50, then the thumb mark is positioned in the middle of the scroll bar.

**Example** 'This example prints a bunch of lines to the viewport, then  
'scrolls back to the top using VScroll.

```
Sub Main()
 ViewportOpen "WM Basic Viewport",100,100,500,200
 For i = 1 to 50
 Print "This will be displayed on line#: " & i
 Next i
 MsgBox "We will now go to the 0% thumb mark poisiton (the top)..."
 VScroll 0
 VScroll 0
 MsgBox "...and here we are!"
 ViewportClose
End Sub
```

**See Also** VLine (statement); VPage (statement).

**Platform(s)** Windows.

## Weekday (function)

---

**Syntax** Weekday(*date*)

**Description** Returns an Integer value representing the day of the week given by date.  
Sunday is 1, Monday is 2, and so on.

The *date* parameter is any expression representing a valid date.

**Example** 'This example gets a date in an input box and displays  
'the day of the week and its name for the date entered.

```
Sub Main()
 Dim a$(7)
 a$(1) = "Sunday"
 a$(2) = "Monday"
 a$(3) = "Tuesday"
 a$(4) = "Wednesday"
 a$(5) = "Thursday"
 a$(6) = "Friday"
 a$(7) = "Saturday"

 Reprompt:
 bd = InputBox$("Please enter your birthday.", "Enter Birthday")
 If Not(IsDate(bd)) Then Goto Reprompt

 dt = DateValue(bd)
 dw = WeekDay(dt)
 MsgBox "You were born on day " & dw & ", which was a " & a$(dw)
End Sub
```

**See Also** Day (function); Minute (function); Second (function); Month (function); Year  
(function); Hour (function); DatePart (function).

**Platform(s)** Windows and Macintosh.



---

## While...Wend (statement)

---

|                                  |                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>                    | While <i>condition</i><br>[ <i>statements</i> ]<br>Wend                                                                                                                                                                                                                                                                                                                                                              |
| <b>Description</b>               | Repeats a statement or group of statements while a condition is True.                                                                                                                                                                                                                                                                                                                                                |
| <b>Comments</b>                  | The condition is initially and then checked at the top of each iteration through the loop.                                                                                                                                                                                                                                                                                                                           |
| <b>Example</b>                   | <p>'This example executes a While loop until the random number generator returns a value of 1.</p> <pre> Sub Main()     x% = 0     count% = 0     While x% &lt;&gt; 1 And count% &lt; 500         x% = Rnd(1)         If count% &gt; 1000 Then             Exit Sub         Else             count% = count% + 1         End If     Wend     MsgBox "The loop executed " &amp; count% &amp; " times." End Sub </pre> |
| <b>See Also</b>                  | Do...Loop (statement); For...Next (statement).                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Platform(s)</b>               | Windows and Macintosh.                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Platform Notes: Windows</b>   | Due to errors in program logic, you can inadvertently create infinite loops in your code. Under Windows, you can break out of infinite loops using Ctrl+Break.                                                                                                                                                                                                                                                       |
| <b>Platform Notes: Macintosh</b> | Due to errors in program logic, you can inadvertently create infinite loops in your code. On the Macintosh, you can break out of infinite loops using Command+Period.                                                                                                                                                                                                                                                |

---

## Width# (statement)

---

|                    |                                                                                       |
|--------------------|---------------------------------------------------------------------------------------|
| <b>Syntax</b>      | Width# <i>filenumber</i> , <i>newwidth</i>                                            |
| <b>Description</b> | Specifies the line width for sequential files opened in either Output or Append mode. |

**Comments** The `Width#` statement requires the following parameters:

| Parameter         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenumber</i> | Integer used by WM Basic to refer to the open file—the number passed to the <code>Open</code> statement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <i>newwidth</i>   | <p>Integer between 0 to 255 inclusive specifying the new width. If <i>newwidth</i> is 0, then no maximum line length is used.</p> <p>When a file is initially opened, there is no limit to line length. This command forces all subsequent output to the specified file to use the specified value as the maximum line length.</p> <p>The <code>Width</code> statement affects output in the following manner: if the column position is greater than 1 and the length of the text to be written to the file causes the column position to exceed the current line width, then the data is written on the next line.</p> <p>The <code>Width</code> statement also affects output of the <code>Print</code> command when used with the <code>Tab</code> and <code>Spc</code> functions.</p> |

**Example**

```
'This statement sets the maximum line width for file number 1 to 80
'columns.

Sub Main()
 Width #1,80
End Sub
```

**See Also** `Print (statement); Print# (statement); Tab (function); Spc (function).`

**Platform(s)** Windows and Macintosh.

---

## **WinActivate (statement)**

---

**Syntax** `WinActivate [window_name$ | window_object] [ ,timeout]`

**Description** Activates the window with the given name or object value.

**Comments** The WinActivate statement requires the following parameters:

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>window_name\$</i> | String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word."<br><br>A hierarchy of windows can be specified by separating each window name with a vertical bar ( ), as in the following example:<br><br><pre>WinActivate "Notepad Find"</pre><br>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find". |
| <i>window_object</i> | HWND object specifying the exact window to activate. This can be used in place of the <i>window_name\$</i> parameter to indicate a specific window to activate.                                                                                                                                                                                                                                                                                                                                                                                                             |
| <i>timeout</i>       | Integer specifying the number of milliseconds for which to attempt activation of the specified window. If not specified (or 0), then only one attempt will be made to activate the window. This value is handy when you are not certain that the window you are attempting to activate has been created.                                                                                                                                                                                                                                                                    |

If *window\_name\$* and *window\_object* are omitted, then no action is performed.

**Example** 'This example runs the clock.exe program by activating the Run File 'dialog box from within Program Manager.

```
Sub Main()
 WinActivate "Program Manager"
 Menu "File.Run"
 WinActivate "Program Manager|Run"
 SendKeys "clock.exe{ENTER}"
End Sub
```

**See Also** AppActivate (statement).

**Platform(s)** Windows.

## WinClose (statement)

**Syntax** WinClose [*window\_name\$* | *window\_object*]

**Description** Closes the given window.

**Comments** The WinClose statement requires the following parameters:

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>window_name\$</i> | String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word."<br><br>A hierarchy of windows can be specified by separating each window name with a vertical bar ( ), as in the following example:<br><br><pre>WinActivate "Notepad Find"</pre><br>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find". |
| <i>window_object</i> | HWND object specifying the exact window to activate. This can be used in place of the <i>window_name\$</i> parameter to indicate a specific window to activate.<br><br>If <i>window_name\$</i> and <i>window_object</i> are omitted, then the window with the focus is closed.<br><br>This command differs from the AppClose command in that this command operates on the current window rather than the current top-level window (or application).                                                                                                                         |

**Example** 'This example closes Microsoft Word if its object reference is found.

```
Sub Main()
 Dim WordHandle As HWND
 Set WordHandle = WinFind("Word")
 If (WordHandle Is Not Nothing) Then WinClose WordHandle
End Sub
```

**See Also** WinFind (function)

**Platform(s)** Windows.

**Platform Notes:** Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.

**Windows**

## WinFind (function)

---

**Syntax** WinFind(*name\$*) As HWND

**Description** Returns an object variable referencing the window having the given name.

**Comments** The *name\$* parameter is specified using the same format as that used by the WinActivate statement.

**Example** 'This example closes Microsoft Word if its object reference is found.

```
Sub Main()
 Dim WordHandle As HWND
 Set WordHandle = WinFind("Word")
 If (WordHandle Is Not Nothing) Then WinClose WordHandle
End Sub
```

**See Also** WinActivate (statement).

**Platform(s)** Windows.

## WinList (statement)

**Syntax** WinList *ArrayOfWindows*()

**Description** Fills the passed array with references to all the top-level windows.

**Comments** The passed array must be declared as an array of HWND objects.

The *ArrayOfWindows* parameter must specify either a zero- or one-dimensional dynamic array or a single-dimensional fixed array. If the array is dynamic, then it will be redimensioned to exactly hold the new number of elements. For fixed arrays, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are unused. A runtime error results if the array is too small to hold the new elements.

After calling this function, use the LBound and UBound functions to determine the new size of the array.

**Examples** 'This example minimizes all top-level windows.

```
Sub Main()
 Dim a() As HWND
 WinList a
 For i = 1 To UBound(a)
 WinMinimize a(i)
 Next i
End Sub
```

**See Also** WinFind (function)

**Platform(s)** Windows.

## WinMaximize (statement)

**Syntax** WinMaximize [*window\_name\$* | *window\_object*]

**Description** Maximizes the given window.

**Comments** The WinMaximize statement requires the following parameters:

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>window_name\$</i> | <p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word."</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar ( ), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".</p> |
| <i>window_object</i> | <p>HWND object specifying the exact window to activate. This can be used in place of the <i>window_name\$</i> parameter to indicate a specific window to activate.</p> <p>If <i>window_name\$</i> and <i>window_object</i> are omitted, then the window with the focus is maximized.</p> <p>This command differs from the AppMaximize command in that this command operates on the current window rather than the current top-level window.</p>                                                                                                                                 |

**Example** 'This example maximizes all top-level windows.

```
Sub Main()
 Dim a() As HWND
 WinList a
 For i = 1 To UBound(a)
 WinMaximize a(i)
 Next i
End Sub
```

**See Also** WinMinimize (statement); WinRestore (statement).

**Platform(s)** Windows.

**Platform Notes:** Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.

**Windows**

---

## WinMinimize (statement)

---

**Syntax** WinMinimize [*window\_name\$* | *window\_object*]

**Description** Minimizes the given window.

**Comments** The WinMinimize statement requires the following parameters:

| Parameter              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>window_name\$</i>   | <p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word."</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar ( ), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".</p> |
| <i>window_object</i>   | <p>HWND object specifying the exact window to activate. This can be used in place of the <i>window_name\$</i> parameter to indicate a specific window to activate.</p> <p>If <i>window_name\$</i> and <i>window_object</i> are omitted, then the window with the focus is minimized.</p> <p>This command differs from the AppMinimize command in that this command operates on the current window rather than the current top-level window.</p>                                                                                                                                 |
| <b>Example</b>         | See example for WinList (statement).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>See Also</b>        | WinMaximize (statement); WinRestore (statement).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Platform(s)</b>     | Windows.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Platform Notes:</b> | Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Windows</b>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

## WinMove (statement)

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <b>Syntax</b>      | WinMove <i>x,y</i> [ <i>window_name\$</i>   <i>window_object</i> ] |
| <b>Description</b> | Moves the given window to the given x,y position.                  |

**Comments** The WinMove statement requires the following parameters:

| Parameter              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>x,y</i>             | Integer coordinates given in twips that specify the new location for the window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <i>window_name\$</i>   | <p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word."</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar ( ), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".</p> |
| <i>window_object</i>   | <p>HWND object specifying the exact window to activate. This can be used in place of the <i>window_name\$</i> parameter to indicate a specific window to activate.</p> <p>If <i>window_name\$</i> and <i>window_object</i> are omitted, then the window with the focus is moved.</p> <p>This command differs from the AppMove command in that this command operates on the current window rather than the current top-level window. When moving child windows, remember that the <i>x</i> and <i>y</i> coordinates are relative to the client area of the parent window.</p>    |
| <b>Example</b>         | <p>'This example moves Program Manager to upper left corner of the screen.</p> <pre>WinMove 0,0,"Program Manager"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>See Also</b>        | WinSize (statement).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Platform(s)</b>     | Windows.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Platform Notes:</b> | Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Windows</b>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

## WinRestore (statement)

---

**Syntax** WinRestore [*window\_name\$* | *window\_object*]

**Description** Restores the specified window to its restore state.



**Comments** Restoring a minimized window restores that window to its screen position before it was minimized. Restoring a maximized window resizes the window to its size previous to maximizing.

The WinRestore statement requires the following parameters:

| Parameter              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>window_name\$</i>   | String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word."<br><br>A hierarchy of windows can be specified by separating each window name with a vertical bar ( ), as in the following example:<br><br><pre>WinActivate "Notepad Find"</pre><br>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find". |
| <i>window_object</i>   | HWND object specifying the exact window to activate. This can be used in place of the <i>window_name\$</i> parameter to indicate a specific window to activate.<br><br>If <i>window_name\$</i> and <i>window_object</i> are omitted, then the window with the focus is restored.<br><br>This command differs from the AppRestore command in that this command operates on the current window rather than the current top-level window.                                                                                                                                      |
| <b>Example</b>         | 'This example minimizes all top-level windows except for Program Manager.<br><br><pre>Sub Main()     Dim a() As HWND     WinList a     For i = 0 To UBound(a)         WinMinimize a(i)     Next i     WinRestore "Program Manager" End Sub</pre>                                                                                                                                                                                                                                                                                                                            |
| <b>See Also</b>        | WinMaximize (statement); WinMinimize (statement).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Platform(s)</b>     | Windows.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Platform Notes:</b> | Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Windows</b>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

## WinSize (statement)

**Syntax** WinSize *width,height* [,*window\_name\$* | *window\_object*]

**Description** Resizes the given window to the specified width and height.

**Comments** The `WinSize` statement requires the following parameters:

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>width,height</i>  | Integer coordinates given in twips that specify the new size of the window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>window_name\$</i> | <p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word."</p> <p>A hierarchy of windows can be specified by separating each window name with a vertical bar (<code> </code>), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".</p> |
| <i>window_object</i> | <p>HWND object specifying the exact window to activate. This can be used in place of the <i>window_name\$</i> parameter to indicate a specific window to activate.</p> <p>If <i>window_name\$</i> and <i>window_object</i> are omitted, then the window with the focus is resized.</p> <p>This command differs from the <code>AppSize</code> command in that this command operates on the current window rather than the current top-level window.</p>                                                                                                                                       |

**Example** 'This example runs and resizes Notepad.

```
Sub Main()
 Dim NotepadApp As HWND
 id = Shell("Notepad.exe")
 set NotepadApp = WinFind("Notepad")
 WinSize 4400,8500,NotepadApp
End Sub
```

**See Also** `WinMove` (statement).

**Platform(s)** Windows.

**Platform Notes:** Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.

**Windows**

---

## Word\$ (function)

**Syntax** `Word$(text$,first[,last])`

**Description** Returns a `String` containing a single word or sequence of words between *first* and *last*.

**Comments** The Word\$ function requires the following parameters:

| Parameter     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>text\$</i> | String from which the sequence of words will be extracted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>first</i>  | Integer specifying the index of the first word in the sequence to return. If <i>last</i> is not specified, then only that word is returned.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <i>last</i>   | Integer specifying the index of the last word in the sequence to return. If <i>last</i> is specified, then all words between <i>first</i> and <i>last</i> will be returned, including all spaces, tabs, and end-of-lines that occur between those words.<br><br>Words are separated by any nonalphanumeric characters such as spaces, tabs, end-of-lines, and punctuation.<br><br>If <i>first</i> is greater than the number of words in <i>text\$</i> , then a zero-length string is returned.<br><br>If <i>last</i> is greater than the number of words in <i>text\$</i> , then all words from <i>first</i> to the end of the text are returned. |

**Example** 'This example finds the name "Stuart" in a string and then extracts two words from the string.

```
Sub Main()
 s$ = "My last name is Williams; Stuart is my surname."
 c$ = Word$(s$,5,6)
 MsgBox "The extracted name is: " & c$
End Sub
```

**See Also** Item\$ (function); ItemCount (function); Line\$ (function); LineCount (function); WordCount (function).

**Platform(s)** Windows and Macintosh.

## WordCount (function)

**Syntax** WordCount(*text\$*)

**Description** Returns an Integer representing the number of words in the specified text.

**Comments** Words are separated by spaces, tabs, and end-of-lines.

**Example** 'This example counts the number of words in a particular string.

```
Sub Main()
 s$ = "My last name is Williams; Stuart is my surname."
 i% = WordCount(s$)
 MsgBox "" & s$ & " has " & i% & " words."
End Sub
```

**See Also** `Item$ (function)`; `ItemCount (function)`; `Line$ (function)`; `LineCount (function)`; `Words$ (function)`.

**Platform(s)** Windows and Macintosh.

### Write# (statement)

**Syntax** Write [#]*filename* [, *expressionlist*]

**Description** Writes a list of expressions to a given sequential file.

**Comments** The file referenced by *filename* must be opened in either Output or Append mode.

The *filenumber* parameter is an Integer used by WM Basic to refer to the open file—the number passed to the Open statement.

The following table summarizes how variables of different types are written:

| Data Type           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Any numeric type    | Written as text. There is no leading space, and the period is always used as the decimal separator.                                                                                                                                                                                                                                                                                                                                                           |
| String              | Written as text, enclosed within quotes.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Empty               | No data is written.                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Null                | Written as #NULL#.                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Boolean             | Written as #TRUE# or #FALSE#.                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Date                | Written using the universal date format:<br><div style="text-align: center;">#YYYY-MM-DD HH:MM:SS#</div>                                                                                                                                                                                                                                                                                                                                                      |
| user-defined errors | Written as #ERROR <i>ErrorNumber</i> #, where <i>ErrorNumber</i> is the value of the user-defined error. The word <code>ERROR</code> is not translated.<br><br>The <code>Write</code> statement outputs variables separated with commas. After writing each expression in the list, <code>Write</code> outputs an end-of-line.<br><br>The <code>Write</code> statement can only be used with files opened in <code>Output</code> or <code>Append</code> mode. |

**Example** 'This example opens a file for sequential write, then writes ten records into the file with the values 10...50. Then the file is closed and reopened for read, and the records are read with the Input statement. The results are displayed in a dialog box.

```
Sub Main()
 Open "test.dat" For Output Access Write As #1
 For x = 1 To 10
 r% = x * 10
 Write #1,x,r%
 Next x
 Close

 Open "test.dat" For Input Access Read As #1
 For x = 1 To 10
 Input #1,a%,b%
 msg = msg & "Record " & a% & ": " & b% & Basic.Eoln$
 Next x

 MsgBox msg
 Close
End Sub
```

**See Also** Open (statement); Put (statement); Print# (statement).

**Platform(s)** Windows and Macintosh.

## WriteIni (statement)

**Syntax** WriteIni *section\$, ItemName\$, value\$[, filename\$]*

**Description** Writes a new value into an ini file.

**Comments** The WriteIni statement requires the following parameters:

| Parameter         | Description                                                                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>section\$</i>  | String specifying the section that contains the desired variables, such as "windows." Section names are specified without the enclosing brackets.                                                   |
| <i>ItemName\$</i> | String specifying which item from within the given section you want to change. If <i>ItemName\$</i> is a zero-length string (""), then the entire section specified by <i>section\$</i> is deleted. |
| <i>value\$</i>    | String specifying the new value for the given item. If <i>value\$</i> is a zero-length string (""), then the item specified by <i>ItemName\$</i> is deleted from the ini file.                      |
| <i>filename\$</i> | String specifying the name of the ini file.                                                                                                                                                         |

**Example** 'This example sets the txt extension to be associated with Notepad.

```
Sub Main()
 WriteIni "Extensions", "txt", "c:\windows\notepad.exe
^.txt", "win.ini"
End Sub
```

**See Also** ReadIni\$ (function); ReadIniSection (statement).

**Platform(s)** Windows.

**Platform** Under Windows, if *filename\$* is not specified, the win.ini file is used.

**Notes:**  
**Windows** If the *filename\$* parameter does not include a path, then this statement looks for ini files in the Windows directory.

---

## Xor (operator)

---

**Syntax** *expression1* Xor *expression2*

**Description** Performs a logical or binary exclusion on two expressions.

**Comments** If both expressions are either Boolean, Boolean variants, or Null variants, then a logical exclusion is performed as follows:

| If the first<br>expression is | and the second<br>expression is | then the<br>result is |
|-------------------------------|---------------------------------|-----------------------|
| True                          | True                            | False                 |
| True                          | False                           | True                  |
| False                         | True                            | True                  |
| False                         | False                           | False                 |

If either expression is Null, then Null is returned.

### Binary Exclusion

If the two expressions are Integer, then a binary exclusion is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long , and a binary exclusion is then performed, returning a Long result.

Binary exclusion forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:

|   |     |   |   |   |                                                                  |
|---|-----|---|---|---|------------------------------------------------------------------|
| 1 | Xor | 1 | = | 0 | Example:<br>5    01101001<br>6    10101010<br><hr/> Xor 11000011 |
| 0 | Xor | 1 | = | 1 |                                                                  |
| 1 | Xor | 0 | = | 1 |                                                                  |
| 0 | Xor | 0 | = | 0 |                                                                  |

**Example** 'This example builds a logic table for the XOR function and displays it.

```
Sub Main()
 For x = -1 To 0
 For y = -1 To 0
 z = x Xor y
 msg = msg & Format(x,"True/False") & " Xor "
 msg = msg & Format(y,"True/False") & " = "
 msg = msg & Format(z,"True/False") & Basic.Eoln$
 Next y
 Next x
 MsgBox msg
End Sub
```

**See Also** Operator Precedence (topic); Or (operator); Eqv (operator); Imp (operator); And (operator).

**Platform(s)** Windows and Macintosh.

## Year (function)

**Syntax** Year(*date*)

**Description** Returns the year of the date encoded in the specified *date* parameter. The value returned is between 100 and 9999 inclusive.

The *date* parameter is any expression representing a valid date.

**Example** 'This example returns the current year in a dialog box.

```
Sub Main()
 tdate$ = Date$
 tyear! = Year(DateValue(tdate$))
 MsgBox "The current year is: " & tyear$
End Sub
```

**See Also** Day (function); Minute (function); Second (function); Month (function); Hour (function); Weekday (function); DatePart (function).

**Platform(s)** Windows and Macintosh.





## Working Model Basic Extensions Reference

This chapter contains an alphabetical listing of objects, methods, and properties that are specifically implemented to operate Working Model.

The reader is strongly encouraged to keep *Working Model User's Manual* at his/her disposal, since most of the following API calls have equivalent implementation in Working Model's graphical user interface. The *Working Model User's Manual* provides far more extensive discussions on features.

### Notations

Italicized portions of headings (e.g., *WMDocument.Body*) indicate type names. Hence to make use of the indicated syntax, you must substitute the name of a variable of that type.

For example, you cannot write:

```
Dim aBody as WMBody
Set aBody = WMDocument.Body(3) ' wrong usage: WMDocument is
a type name
```

Instead, you must declare a variable of type *WMDocument* as follows:

```
Dim aDoc as WMDocument
Dim aBody as WMBody
Set aDoc = WM.ActiveDocument
Set aBody = aDoc.Body(3) ' correct
```

Note that in the last line of the above example, *WMDocument* is replaced with *aDoc*, a variable of the type *WMDocument* in order to invoke the *Body* method.

### Collection (topic)

In Working Model Basic, some objects are called a *Collection*, because they contain a set of objects of a particular type. The following objects are Collections defined in Working Model Basic.

| Object            | Description                                                          |
|-------------------|----------------------------------------------------------------------|
| WM.Documents      | Collection of Working Model documents, or <i>WMDocument</i> objects. |
| WMDocument.Bodies | Collection of all <i>WMBody</i> objects in the document.             |

## 518 Working Model Basic User's Manual

---

Each Collection object has several common names for its properties and methods, although the exact syntax, return types, and their actions differ depending on the type of objects that the Collection represents.

Common properties and methods are as follows:

---

| Method/Property | Description |
|-----------------|-------------|
|-----------------|-------------|

---

|                  |                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------|
| Count (property) | The Integer property containing the current number of objects that the Collection represents. |
|------------------|-----------------------------------------------------------------------------------------------|

|               |                                                                                                                                                                                                                                                               |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Item (method) | Returns the specific object within the Collection. For example, <code>WMDocument.Bodies.Item</code> returns a <code>WMBody</code> object.<br><br>Please refer to individual Collection objects for detailed information on definitions, syntax, and examples. |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**See Also** `WM.Documents` (property), `WMDocument.Selection` (property)

## WM (constant)

---

|                    |                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | The constant <code>WM</code> represents the Working Model application itself. <code>WM</code> is an object with the following methods and properties. Please see individual sections for detailed information on syntax, return values, and examples. |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

---

| Method/Property | Description |
|-----------------|-------------|
|-----------------|-------------|

---

|                             |                                          |
|-----------------------------|------------------------------------------|
| <code>ActiveDocument</code> | Currently active Working Model document. |
|-----------------------------|------------------------------------------|

|                             |                                           |
|-----------------------------|-------------------------------------------|
| <code>DeleteMenuItem</code> | Deletes a menu item from the Script menu. |
|-----------------------------|-------------------------------------------|

|                        |                                 |
|------------------------|---------------------------------|
| <code>Documents</code> | Set of documents that are open. |
|------------------------|---------------------------------|

|                             |                                                                      |
|-----------------------------|----------------------------------------------------------------------|
| <code>EnableMenuItem</code> | Enables or disables the user-definable menu item in the Script menu. |
|-----------------------------|----------------------------------------------------------------------|

|                          |                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------|
| <code>GetMenuItem</code> | Returns a String containing the name of the specified menu item in the Script menu. |
|--------------------------|-------------------------------------------------------------------------------------|

|                             |                                                                                                 |
|-----------------------------|-------------------------------------------------------------------------------------------------|
| <code>InsertMenuItem</code> | Adds a new menu item in the Script menu and associates the item with the specified script file. |
|-----------------------------|-------------------------------------------------------------------------------------------------|

|                             |                                                               |
|-----------------------------|---------------------------------------------------------------|
| <code>LoadWMBLibrary</code> | Loads a file containing subroutines for use in other scripts. |
|-----------------------------|---------------------------------------------------------------|

|                  |                                                |
|------------------|------------------------------------------------|
| <code>New</code> | Opens a new (untitled) Working Model document. |
|------------------|------------------------------------------------|

|                   |                                           |
|-------------------|-------------------------------------------|
| <code>Open</code> | Opens an existing Working Model document. |
|-------------------|-------------------------------------------|

|                        |                                             |
|------------------------|---------------------------------------------|
| <code>RunScript</code> | Executes a script/tool written in WM Basic. |
|------------------------|---------------------------------------------|

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| <code>ShowAppearanceWindow</code> | Opens/closes the Appearance window. |
|-----------------------------------|-------------------------------------|

|                                 |                                   |
|---------------------------------|-----------------------------------|
| <code>ShowGeometryWindow</code> | Opens/closes the Geometry window. |
|---------------------------------|-----------------------------------|

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| <code>ShowPropertiesWindow</code> | Opens/closes the Properties window. |
|-----------------------------------|-------------------------------------|

**UnloadWMBLibrary** Unloads a file containing subroutines used in other scripts.

**Version** Version number of Working Model.

**Example** (See WM.ActiveDocument)

**See Also** WM.ActiveDocument (property), WM.Documents (property), WM.New (method), WM.Open (method), WM.Version (property), WM.Quit (method), WM.Save (method), WM.SaveAs (method).

---

## WM.ActiveDocument (property)

---

**Syntax** WM.ActiveDocument

**Description** A WMDocument object which is currently active in the Working Model application. You can set or get the WM.ActiveDocument property.

**Comments** Working Model can open and simulate several documents at a time, and you need to make a document active in order to bring it to the foreground. WM.ActiveDocument returns an error if the given document is invalid. If no document is currently open, WM.ActiveDocument returns an error code.

**Example**

```
Sub Main()
 'opens a new document in the back layer.
 Dim Doc1 as WMDocument
 Dim Doc2 as WMDocument
 Set Doc1 = WM.ActiveDocument
 Set Doc2 = WM.New() 'opens a new document (becomes active).
 If Doc1 is not Nothing then
 Set WM.ActiveDocument = Doc1 'resurrects the previously
 active document
 End If
End Sub
```

**See Also** WMDocument (object), WM.ActiveDocument (statement)

---

## WM.DeleteMenuItem (method)

---

**Syntax** WM.DeleteMenuItem *Index*

**Description** Deletes a menu item from the Script menu.

**Comments** The WM.DeleteMenuItem method removes a menu item from Working Model's Script menu (located in the menu bar). The first two items in the Script menu, Run and Edit, are part of Working Model's standard menu and cannot be deleted. Any menu item added by WM.InsertMenuItem

can be removed using the `WM.DeleteMenuItem` method.

The `WM.DeleteMenuItem` method takes the following parameter:

| Parameter    | Description                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Index</i> | <p>An Integer. The number (&gt; 0) specifies the location of the menu item to be removed.</p> <p>For example, if three menu items exist below <code>Editor</code>, then <i>Index</i> can be between 1 and 3 (inclusive). Setting <i>Index</i> = 2 would remove the menu item in the middle, for instance.</p> <p>If <i>Index</i> is out of range, <code>DeleteMenuItem</code> neither removes any item nor generates an error code.</p> |

**Example**

```
Sub Main()
 ' Adds three menu items "Hop", "Step", and "Jump"
 ' in that order. The pathname is shown in Windows format.
 WM.InsertMenuItem 1, "Hop", "C:\scripts\hop.wbs"
 WM.InsertMenuItem 2, "Step", "C:\scripts\step.wbs"
 WM.InsertMenuItem 3, "Jump", "C:\scripts\jump.wbs"
 ' Immediately deletes the "Step" menu.
 WM.DeleteMenuItem 2
End Sub
```

**See Also** `WM.InsertMenuItem` (method), `WM.EnableMenuItem` (method)

---

## WM.Documents (property)

---

|                    |                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WM.Documents</code>                                                                                                                      |
| <b>Description</b> | Returns a collection object of currently open Working Model documents.                                                                         |
| <b>Comments</b>    | <p><code>WM.Documents</code> is a read-only property.</p> <p>The <code>WM.Documents</code> property has the following property and method.</p> |

---

| Method/Property | Description |
|-----------------|-------------|
|-----------------|-------------|

---

|                       |                                                                                                                   |
|-----------------------|-------------------------------------------------------------------------------------------------------------------|
| <code>Count</code>    | An Integer property containing the number of documents in <code>WM.Documents</code> (= number of open documents). |
| <code>Item(id)</code> | A method which returns the <code>WMDocument</code> object specified by the index <i>id</i> (Integer).             |

**Example**

```
Sub Main()
```

```

' Shows the name of the third document. Does nothing
' if no more than 2 documents are currently open

Dim WM1 as WMDocument

if WM.Documents.Count > 2 then
 Set WM1 = WM.Documents.Item(3)
 MsgBox "Document ID 3 is " + WM1.Name
end if

End Sub

```

**See Also** WMDocument (object), WM (constant)

## WM.EnableMenuItem (method)

**Syntax** WM.EnableMenuItem *Index,EnableFlag*

**Description** Enables or disables a menu item in the Script menu.

**Comments** The WM.EnableMenuItem method enables or disables a user-definable item from Working Model's Script menu (located in the menu bar). The first two items in the Script menu, Run and Edit, are part of Working Model's standard menu and cannot be altered. Any menu item added by WM.InsertMenuItem can be enabled or disabled using the WM.EnableMenuItem method.

The disabled menu item remains in the Script menu but appears dimmed (shown in light-gray). The item remains as such until it is enabled by the EnableMenuItem method. While the menu item is disabled, the user cannot choose it and is prohibited from invoking the script or tool associated with the menu item through the Script menu. The user can still invoke the script using the Run menu (as long as she or he is aware of the filename and path).

The WM.EnableMenuItem method takes the following parameters:

| Parameter         | Description                                                                                                                                                                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Index</i>      | <p>An Integer. The number (&gt; 0) specifies the location of the menu item to be enabled or disabled.</p> <p>For example, if three menu items exist below Editor, then the <i>Index</i> can be between 1 and 3 (inclusive). Setting <i>Index</i> = 2 would enable or disable the menu item in the middle, for instance.</p> |
| <i>EnableFlag</i> | <p>A Boolean specifying whether the menu item is to be enabled or disabled. When EnableFlag is set to True, the menu item will be enabled; if False, then the menu item will be disabled.</p>                                                                                                                               |

If *Index* is out of range, `EnableMenuItem` neither generates an error nor performs any action.

To remove a menu item that was inserted by `EnableMenuItem`, use `WM.DeleteMenuItem` method.

**Example**

```
Sub Main()
 ' Adds three menu items "Hop", "Step", and "Jump"
 ' in that order, and disables Hop and Step menu items.
 ' The pathname is shown in Windows format.
 WM.InsertMenuItem 1, "Hop", "C:\scripts\hop.wbs"
 WM.InsertMenuItem 2, "Step", "C:\scripts\step.wbs"
 WM.InsertMenuItem 3, "Jump", "C:\scripts\jump.wbs"
 ' Disable Hop and Step menus.
 WM.EnableMenuItem 1, False ' corresponds to Hop menu item
 WM.EnableMenuItem 2, False ' corresponds to Step menu item
End Sub
```

**See Also** `WM.DeleteMenuItem` (method), `WM.InsertMenuItem` (method)

---

## WM.GetMenuItem (method)

---

**Syntax** `WM.GetMenuItem(Index [, PathString])`

**Description** Returns a `String` containing the name of the specified menu item in the Script menu.

**Comments** The `WM.GetMenuItem` method returns the menu name of the specified menu item from Working Model's Script menu (located in the menu bar). Further, you can obtain the pathname of the script that is invoked by the menu item. (The pathname is originally set by the `WM.InsertMenuItem` method.)

The first two items in the Script menu, `Run` and `Edit`, are part of Working Model's standard menu and cannot be accessed using `WM.GetMenuItem`. Any menu item added by `WM.InsertMenuItem` can be accessed using the `WM.GetMenuItem` method.

The `WM.GetMenuItem` method takes the following parameters:

| Parameter    | Description                                                                                                                                                                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Index</i> | An <code>Integer</code> . The number (> 0) specifies the location of the menu item. For example, if three menu items exist below <code>Edit</code> in the Script menu, then <i>Index</i> can be between 1 and 3 (inclusive). Setting <i>Index</i> = 2 would |

access the menu item in the middle, for instance.

*PathString*

An optional `String`. This parameter receives the pathname of the script that is originally set by the `WM.InsertMenuItem` method.

If *Index* is out of range, `WM.GetMenuItem` neither generates an error nor performs any action.

If you choose to replace the script/tool invoked by a menu item, you need to delete the menu item first (using `WM.DeleteMenuItem`) and add the menu item again using `WM.InsertMenuItem`, with the new script pathname as its parameter.

The script specified by *PathString* can be either object code or source code. See Chapter 1 for more information on object code and source code file types.

**Example**

```
Sub Main()
 Dim Path as String
 Dim MenuName as String

 ' Adds three menu items "Hop", "Step", and "Jump"
 ' in that order. The pathname is shown in Windows format.
 WM.InsertMenuItem 1, "Hop", "C:\scripts\hop.wbs"
 WM.InsertMenuItem 2, "Step", "C:\scripts\step.wbs"
 WM.InsertMenuItem 3, "Jump", "C:\scripts\jump.wbs"

 ' Report pathname for the Jump menu item.
 MenuName = WM.GetMenuItem(3, Path)

 MsgBox "Menu "+MenuName+" invokes the file "+Path
End Sub
```

**See Also** `WM.InsertMenuItem (method)`, `WM.DeleteMenuItem (method)`

## WM.InsertMenuItem (method)

**Syntax** `WM.InsertMenuItem Index, MenuName, ScriptFileName`

**Description** Adds a new menu item in the Script menu, and associates the menu item with the specified script file.

**Comments** The `InsertMenuItem` method adds a new item in Working Model's Script menu located in the menu bar. At the same time, the specified script is associated with the new menu item. From then on, the user can simply choose the menu item and invoke the script or tool written in WM Basic. The purpose of this method is to provide quick access to pre-written scripts by seamlessly integrating external scripts into Working Model's

user interface.

The `WM.InsertMenuItem` method takes the following parameters:

| Parameter             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Index</i>          | An Integer specifying where the menu item will be added. The number (> 0) specifies the order in which the menu items appear. The number <i>Index</i> must specify where the menu is to be located.<br><br>For example, the first addition should be <i>Index</i> = 1. For the second time around, if you wish to add another menu item <i>above</i> the first, then set <i>Index</i> = 1. If you wish to add it <i>below</i> the first, then set <i>Index</i> = 2.                                                                                                                                                                                                                  |
| <i>MenuName</i>       | A String specifying how the menu item appears in the Script menu.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>ScriptFileName</i> | A String specifying the script or tool that is to be invoked by choosing the menu item. Full pathname is required.<br><br>If <i>Index</i> is out of range, <code>InsertMenuItem</code> does not add any new item. If <i>ScriptFileName</i> is not found in the file system, <code>InsertMenuItem</code> still adds the menu item. In either case, no error code is generated. Use the <code>FileExists</code> call to verify the existence of the script file, if so desired.<br><br>After a menu item is added, you can disable or enable the menu item using <code>WM.EnableMenuItem</code> method. You can also remove the menu item using <code>WM.DeleteMenuItem</code> method. |

**Example** (See `WM.EnableMenuItem`)

**See Also** `WM.DeleteMenuItem` (method), `WM.EnableMenuItem` (method), `FileExists` (function)

---

## WM.LoadWMBLibrary (method)

---

**Syntax** `WM.LoadWMBLibrary ScriptFileName`

**Description** Loads a scripting code module.

**Comments** The `LoadWMBLibrary` method loads a library of WM Basic code so that it becomes available to the user. In order to invoke functions and subroutines implemented in a library code module, you have to use the `LoadWMBLibrary` method to load it first.

The `WM.LoadWMBLibrary` method takes the following parameters:

| Parameter             | Description                                                                                                                                                                                                         |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ScriptFileName</i> | A String specifying the filename of the code module.<br><br>The file could be either a source script file or a compiled script file. If you loaded a source file, you can use the debugger to trace into the source |



code of the library.

---

**NOTE:** After a code module is loaded with `WM.LoadWMBLibrary`, the module will stay in memory until you explicitly unload it using `WM.UnloadWMBLibrary` or quit Working Model. If you make a change in the module, and if the module is already loaded, you must unload the module and reload it to make the change effective.

---

In order to use memory efficiently, you should consider using `WM.UnloadWMBLibrary` method, which unloads the code module.

If you wish to run a complete script or tool (i.e., script or tool that can run on its own), you should use the `WM.RunScript` method.

**Example** ' Suppose the following code is saved as "c:\testing\mod1.wbs".

```
Function TimesTwo(a as Integer)
 ' Defines the function TimesTwo
 TimesTwo = a * 2
End Function

' The following code (the main module) makes a call.
Sub Main()
 Dim I as Integer
 WM.LoadWMBLibrary "c:\testing\mod1.wbs"
 I = 2
 MsgBox str$(TimesTwo(I)) ' Calls the function TimesTwo
 WM.UnloadWMBLibrary "c:\testing\mod1.wbs"
End Sub
```

**See Also** `WM.UnloadWMBLibrary` (method), `WM.DeleteMenuItem` (method), `WM.EnableMenuItem` (method), `FileExists` (function), `WM.RunScript` (method)

## WM.New (method)

---

**Syntax** `WM.New()`

**Description** Opens a new (untitled) Working Model document and returns the corresponding `WMDocument` object.

**Comments** `WM.ActiveDocument` will point to the new document.

**Example**

```
Sub Main()
 Dim WM1 as WMDocument
```

```
Set WM1 = WM.New() 'just opened a new document
End Sub
```

**See Also** WMDocument (object)

---

## WM.Open (method)

---

**Syntax** WM.Open(*FileName*)

**Description** Opens the specified Working Model document and returns the corresponding WMDocument object.

**Comments** The String parameter *FileName* is the name of the Working Model file you wish to open. The parameter can contain pathnames.

WM.ActiveDocument will point to the opened document.

**Example**

```
Sub Main()
 Dim WM1 as WMDocument
 If FileExists("demol.wm") then
 Set WM1 = WM.Open("demol.wm")
 End If
End Sub
```

**See Also** WMApplication (object), WMDocument (object)

---

## WM.RunScript (method)

---

**Syntax** WM.RunScript *FileName*

**Description** Executes a script written in Working Model Basic.

**Comments** The WM.RunScript method invokes and executes the specified script or tool written in Working Model Basic.

The WM.RunScript method takes the following parameter:

| Parameter       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>FileName</i> | <p>A String containing the full pathname of the file/script to be executed (<i>target script</i>).</p> <p>If the target script does not exist, WM.RunScript neither generates an error nor performs any action. Use the FileExists call to verify the existence of the target script to make your code robust.</p> <p>The target script needs to be a <i>complete program</i> (a program that can be executed on its own). If you wish to invoke only functions or subroutines defined in a target script, consider using the WM.LoadWMBLibrary method.</p> |

The WM Basic Debugger cannot trace into the target script. If you wish to debug the target script, you should run the target script on its own.

Otherwise, you can turn the target script into a code module; in that case, you would need to rename `Main()` module to something else, such as `Main_Dummy()`. You can use `WM.LoadWMBLibrary` method to load the module, and call the subroutine `Main_Dummy()`. The debugger will be able to step into the target script code.

If you wish to have quick access to scripts or tools that you would like to run frequently, you may consider using `WM.InsertMenuItem`, which adds specified scripts or tools as menu items in the Script menu.

**Example**

```
Sub Main()
 ' Executes a Script named "magic.wbs". Returns an
 ' error message if no such file is found.

 Dim S as String
 If Basic.OS = ebWin16 then
 S = "c:\scripts\magic.wbs"
 ElseIf Basic.OS = ebMacintosh then
 S = "Macintosh HD:Scripts:magic.wbs"
 Else
 S = "" ' Null string
 End If
 If FileExists(S) then
 WM.RunScript S
 Else
 MsgBox "Tool "+S+" cannot be found."
 End If
End Sub
```

**See Also** `WM.InsertMenuItem` (method), `FileExists` (Function),  
`WM.LoadWMBLibrary` (method)

---

## WM.UnloadWMBLibrary (method)

---

- Syntax** `WM.UnloadWMBLibrary ScriptFileName`
- Description** Unloads a scripting code module.
- Comments** The `UnloadWMBLibrary` method unloads a library of WM Basic code module from memory. The code module must be loaded first (using the

## 528 Working Model Basic User's Manual

---

WM.LoadWMBLibrary method) before being unloaded.

By unloading a code module that may not be used frequently, Working Model can maintain sufficient memory space to run. You can always load the code module back when necessary.

The WM.UnloadWMBLibrary method takes the following parameters:

| Parameter             | Description                                                                                                                      |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>ScriptFileName</i> | A String specifying the filename of the code module.<br>The file could be either a source script file or a compiled script file. |
| <b>Example</b>        | (See WM.LoadWMBLibrary.)                                                                                                         |
| <b>See Also</b>       | WM.LoadWMBLibrary (method), WM.DeleteMenuItem (method),<br>WM.EnableMenuItem (method), FileExists (function)                     |

## WM.ShowAppearanceWindow (property)

---

|                    |                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WM.ShowAppearanceWindow                                                                                                                                                           |
| <b>Description</b> | A Boolean property to specify whether the Appearance window is open.                                                                                                              |
| <b>Comments</b>    | When the ShowAppearanceWindow property is set to True, the Appearance window opens.                                                                                               |
| <b>Example</b>     | <pre>Sub Main()<br/>    ' Opens the Appearance window<br/>    If WM.ShowAppearanceWindow = False Then<br/>        WM.ShowAppearanceWindow = True<br/>    End If<br/>End Sub</pre> |
| <b>See Also</b>    | WM (constant), WM.ShowPropertiesWindow (property),<br>WM.ShowGeometryWindow (property)                                                                                            |

## WM.ShowGeometryWindow (property)

---

|                    |                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WM.ShowGeometryWindow                                                                                   |
| <b>Description</b> | A Boolean property to specify whether the Geometry window is open.                                      |
| <b>Comments</b>    | When the ShowGeometryWindow property is set to True, the Geometry window opens.                         |
| <b>Example</b>     | <pre>Sub Main()<br/>    ' Opens the Geometry window<br/>    If WM.ShowGeometryWindow = False Then</pre> |

```
WM.ShowGeometryWindow = True
```

```
End If
```

```
End Sub
```

**See Also** WM (constant), WM.ShowPropertiesWindow (property),  
WM.ShowGeometryWindow (property)

---

## WM.ShowPropertiesWindow (property)

---

**Syntax** WM.ShowPropertiesWindow

**Description** A Boolean property to specify whether the Properties window is open.

**Comments** When the ShowPropertiesWindow property is set to True, the Properties window opens.

**Example**

```
Sub Main()
 ' Opens the Properties window
 If WM.ShowPropertiesWindow = False Then
 WM.ShowPropertiesWindow = True
 End If
End Sub
```

**See Also** WM (constant), WM.ShowPropertiesWindow (property),  
WM.ShowGeometryWindow (property)

---

## WM.Version (property)

---

**Syntax** WM.Version

**Description** Returns a String indicating the version number of the Working Model application.

**Comments** WM.Version is read-only.

**Example**

```
Sub Main()
 MsgBox "Running Working Model ver. " + WM.Version
End Sub
```

**See Also** WM (topic)

---

## WMBody (object)

---

**Syntax** WMBody

**Description** An object which provides an interface to rigid bodies used in Working Model simulations.

## 530 Working Model Basic User's Manual

**Comments** To create a `WMBody` object in a document, use the `NewBody` method of the `WMDocument` object.

A `WMBody` object has properties that are different depending on the `Kind` of the object. Shown below are properties common to all `WMBody` objects. Please refer to the *Working Model User's Manual* for detailed discussions on these properties.

---

**Note:** `WMBody` object also has properties available in `WMObject` objects. Please see the section on `WMObject` for details.

---

| Property                          | Description                                                                                                                               |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Kind</code>                 | (String) type of the body ( <code>Circle</code> , <code>Rectangle</code> , or <code>Polygon</code> ) (read-only).                         |
| <code>PX</code> , <code>PY</code> | ( <code>WMCell</code> ) x- and y-positions of the body's FOR.                                                                             |
| <code>PR</code>                   | ( <code>WMCell</code> ) orientation of the body.                                                                                          |
| <code>VX</code> , <code>VY</code> | ( <code>WMCell</code> ) x- and y-velocities of the body's FOR.                                                                            |
| <code>VR</code>                   | ( <code>WMCell</code> ) angular velocity of the body.                                                                                     |
| <code>COMOffsetX</code>           | ( <code>WMCell</code> ) x-offset of the body's center of mass from the FOR.                                                               |
| <code>COMOffsetY</code>           | ( <code>WMCell</code> ) y-offset of the body's center of mass from the FOR.                                                               |
| <code>Mass</code>                 | ( <code>WMCell</code> ) mass of the body. Changing <code>Mass</code> affects <code>Density</code> accordingly, as the body area is fixed. |
| <code>Area</code>                 | (double) area of the body (read-only).                                                                                                    |
| <code>Density</code>              | (double) density of the body. Changing <code>Density</code> affects <code>Mass</code> accordingly, as the body area is fixed (read-only). |
| <code>Moment</code>               | ( <code>WMCell</code> ) mass moment of inertia of the body.                                                                               |
| <code>Charge</code>               | ( <code>WMCell</code> ) electric charge of the body.                                                                                      |
| <code>Elasticity</code>           | ( <code>WMCell</code> ) elasticity of the body.                                                                                           |
| <code>StaticFriction</code>       | ( <code>WMCell</code> ) static friction coefficient of the body.                                                                          |
| <code>KineticFriction</code>      | ( <code>WMCell</code> ) kinetic friction coefficient of the body.                                                                         |
| <code>ShowCenterOfMass</code>     | (Boolean) 1 if shown, 0 otherwise.                                                                                                        |
| <code>ShowCharge</code>           | (Boolean) 1 if shown, 0 otherwise.                                                                                                        |
|                                   | Further, depending on the <code>Kind</code> , a <code>WMBody</code> object has the following properties.                                  |
|                                   | <b>For <code>Kind</code> = "Rectangle" and "Square":</b>                                                                                  |
| Property                          | Description                                                                                                                               |

|                                                                                                                         |                                    |
|-------------------------------------------------------------------------------------------------------------------------|------------------------------------|
| Width                                                                                                                   | (WMCCell) width of the rectangle.  |
| Height                                                                                                                  | (WMCCell) height of the rectangle. |
| For Square, Width and Height are always equal. Any modification to either property will automatically modify the other. |                                    |

**For kind = "Circle":**

| Property | Description                     |
|----------|---------------------------------|
| Radius   | (WMCCell) radius of the circle. |

**For kind = "Polygon":**

| Property/Method | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VertexCount     | (Integer) the number of vertices that the polygon consists of.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| GetVertex       | <p>Given an index (Integer) and two parameters (Double), this method fills the parameters <math>x</math> and <math>y</math> with the <math>(x, y)</math> coordinates of the specified vertex.</p> <p>Syntax: <code>Body.GetVertex n, x, y</code></p> <p>where <math>n</math> is the index of the vertex, and <math>x</math> and <math>y</math> are parameters of type Double, to be assigned to the coordinates of the vertex <math>n</math> upon return from the method. The <math>(x, y)</math> coordinates are given in terms of the polygon's frame of reference (FOR).</p>        |
| AddVertex       | <p>Given an index (Integer) and <math>x, y</math> coordinates, this method adds the vertex to the polygon at the specified index.</p> <p>Syntax: <code>Body.AddVertex n, x, y</code></p> <p>where <math>n</math> is the index at which the new vertex <math>(x, y)</math> is added. The rest of the vertices will be pushed down in the list. The <math>(x, y)</math> coordinates are given in terms of the global coordinates.</p>                                                                                                                                                    |
| DeleteVertex    | <p>Given an index, this method deletes the vertex from the polygon.</p> <p>Syntax: <code>Body.DeleteVertex n</code></p> <p>where <math>n</math> is the index of the vertex to be deleted, and <code>Body</code> is assumed to be a <code>WMBody</code> object. The rest of the vertices will be pushed up in the list.</p> <p>Because a polygon must have at least 3 vertices, <code>DeleteVertex</code> fails if <code>VertexCount = 3</code>. If you want to replace one of the three remaining vertices, you must first insert the replacement vertex, then delete the old one.</p> |

**Example**

```
Sub Main()
 'Creates a circle and a polygon
 Dim CircleBody as WMBody
```

```
Dim PolyBody as WMBody

Dim Doc as WMDocument

Set Doc = WM.ActiveDocument

Set CircleBody = Doc.NewBody("Circle")

CircleBody.PX.Value = 1.0: CircleBody.PY.Value = 2.5

CircleBody.Radius.Value = 1.5

Set PolyBody = Doc.NewBody("polygon")

PolyBody.PX.Value = 4.0: PolyBody.PY.Value = 2.5

' Form the square-shaped polygon

PolyBody.AddVertex 1, 0.5, 0.5

PolyBody.AddVertex 2, 0.5, -0.5

PolyBody.AddVertex 3, -0.5, 0.5

PolyBody.AddVertex 4, -0.5, -0.5

PolyBody.DeleteVertex 7 ' delete three default vertices

PolyBody.DeleteVertex 6

PolyBody.DeleteVertex 5

' Delete one of the vertices and turn it into a right triangle

PolyBody.DeleteVertex 3 'Delete one of the vertices

End Sub
```

**See Also** WMDocument.NewBody (method), WMConstraint (object), WMPoint (object), WMCell (object)

---

## WMCell (object)

---

**Syntax** WMCell

**Description** WMCell objects are formula cells used in Working Model simulations.

**Comments** WMCell is a placeholder for the properties available in Working Model objects. For example, a WMBody object has properties PX and PY, which represent the position of the body. You can enter in these properties not only numerical values but Working Model formulas (such as body[5].v.x). These properties, PX and PY, are instances of WMCell objects.

A WMCell object has the following properties.

| Property | Description                                                              |
|----------|--------------------------------------------------------------------------|
| Formula  | A String that contains the formula string. The usage of formula language |



is discussed in the *Working Model User's Manual*.

**Value** A Double that contains a numerical value. If the Formula string is not empty, the Value property holds the result of the evaluation of the Formula string above.

To assign a formula language expression to a WMCell object, make sure to bracket the expression with double quotation marks ("). For example, in order to set a position, initial velocity, and a dimension of a rectangle body called Rect, you can assign numerical values and formulas as follows:

```
Rect.PX.Formula = "time + 2" ' x-position
Rect.PY.Value = 15 ' y-position
Rect.Width.Value = 8 ' width of the rectangle
Rect.Height.Formula = "Input[5]" ' height of the rectangle
```

As soon as one of the WMCell fields is defined, the other follows suit. For example, if you define a valid formula expression in the Formula property of a WMCell object, its Value property will hold the result of the evaluation of the formula thereafter. Conversely, if you define a numerical value in the Value property of the WMCell object, the Formula property will hold an empty string.

When the Formula property is specified, its value is evaluated at every frame of the simulation step, including frame 0. The value of *t* or *time* is evaluated to 0 at frame 0.

For performance reasons, when a WMCell object is modified, the appearance on the screen does not update until the script is terminated or the Update method is invoked (although internal computations use correct values). See WMDocument.Update for more information.

---

**Note:** WMCell objects are always associated with properties of other objects, such as WMBody and WMConstraint. Declaring a WMCell object means nothing more than declaring a “pointer” and does not allocate the actual object. Therefore, you cannot use a WMCell object as a placeholder for expressions and variables. For example, you cannot type:

```
Dim MyCell as WMCell
MyCell.Formula = "2.5 + sin(t)" ' you will get an error here
```

because the object MyCell is still only a pointer and has no substance. Meanwhile, the following usage is valid:

```
Dim MyBody as WMBody
Dim MyCell as WMCell
Set MyBody = WM.ActiveDocument.NewBody("square")
```

```
Set MyCell = MyBody.PX ' correct usage
```

because `MyCell` now points to the physical object, `MyBody.PX`, or the x-position of a square `WMOBJECT`.

---

**Example**

```
Sub Main()
 ' Creates a rectangle and an input, and
 ' links the two.

 Dim D as WMDocument
 Dim B as WMBody
 Dim I as WMInput
 Set D = WM.ActiveDocument
 Set B = D.NewBody("Rectangle")
 Set I = D.NewInput()

 ' Moves the slider. Note X and Y are simply
 ' integers and not WMCell objects.
 I.X = 100
 I.Y = 50
 I.Value = 0.5

 ' Defines the rectangle's initial position and initial velocity
 B.PX.Value = 1.0
 B.PY.Value = 2.0
 B.VX.Formula = "Input["+str$(I.ID)+"]"

 ' Defines the rectangle's dimension
 B.Width.Value = 0.4
 B.Height.Formula = "body["+str$(B.ID)+"].width*2"

End Sub
```

**See Also** `WMBody (object)`, `WMPoint (object)`, `WMConstraint (object)`, `WMInput (object)`, `WMOutput (object)`, `WMDocument (object)`, `WMDocument.Update (object)`

---

## WMConstraint (object)

---

**Syntax** `WMConstraint`

**Description** An object which provides an interface to constraints used in Working Model simulations.

**Comments** To create a new `WMConstraint` object in a Working Model document, use the `NewConstraint` method of the `WMDocument` object.

All WMConstraint objects have the following methods and properties. The list shows the properties and methods applicable to each Kind. Note that only Kind, ActiveWhen, and AlwaysActive are the common properties for all WMConstraint objects. Please refer to individual sections for these methods and properties for more information.

**Note:** WMConstraint object also has properties available in WMOBJECT objects. Please see the section on WMOBJECT for details.

| Constraint Kind | Applicable Properties and Methods                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------|
| Pin, SquarePin  | Kind, ActiveWhen, AlwaysActive, Point, PointCount                                                                               |
| Spring          | Kind, ActiveWhen, AlwaysActive, Point, K, Exponent, Length, CurrentLength, PointCount                                           |
| Damper          | Kind, ActiveWhen, AlwaysActive, Point, K, Exponent, PointCount                                                                  |
| SpringDamper    | Kind, ActiveWhen, AlwaysActive, K, DamperK, Length, CurrentLength, Point, PointCount                                            |
| Rspring         | Kind, ActiveWhen, AlwaysActive, K, Exponent, Rotation, CurrentRotation, Point, PointCount                                       |
| Rdamper         | Kind, ActiveWhen, AlwaysActive, K, Exponent, Point, PointCount                                                                  |
| Slot            | Kind, ActiveWhen, AlwaysActive, Point, PointCount                                                                               |
| CurvedSlot      | Kind, ActiveWhen, AlwaysActive, Point, VertexCount, GetVertex, AddVertex, DeleteVertex, ClosedSlot, PointCount                  |
| Rod             | Kind, ActiveWhen, AlwaysActive, Point, Length, CurrentLength, PointCount                                                        |
| Separator, Rope | Kind, ActiveWhen, AlwaysActive, Point, Elasticity, Length, CurrentLength, PointCount                                            |
| Force           | Kind, ActiveWhen, AlwaysActive, Point, FX, FY, FR, Ftheta, RotateWithBody, PointCount                                           |
| Torque          | Kind, ActiveWhen, AlwaysActive, Point, Torque, PointCount                                                                       |
| Actuator        | Kind, ActiveWhen, Field, AlwaysActive, Point, ActuatorType, PointCount                                                          |
| Motor           | Kind, ActiveWhen, Field, AlwaysActive, Point, MotorType, PointCount                                                             |
| Pulley          | AppendPoint, PointCount, Point                                                                                                  |
| Gear            | Kind, ActiveWhen, AlwaysActive, GearRatio, AutoComputeGearRatio, RodActive, RodAlwaysActive, Internal, InternalBody, PointCount |

### Example

```
Sub Main()
 ' Creates a box, spring, and attaches one endpoint of
```

```
' the spring to Box.

Dim Doc as WMDocument : Set Doc = WM.Activedocument
Dim Box as WMBody
Set Box = Doc.NewBody("square") : Box.Width.Value = 1.0
Dim MySpring as WMConstraint
Set MySpring = Doc.NewConstraint("spring")
Set MySpring.Point(1).Body = Box ' attaches an endpoint to Box
' point coordinates are relative to Box, since the point is
' already attached to it.
MySpring.Point(1).PX.Value = 0.2
MySpring.Point(1).PY.Value = 0.5
' Set the position of the other endpoint to (3, 0) (global)
MySpring.Point(2).PX.Value = 3.0
MySpring.Point(2).PY.Value = 0.0

End Sub
```

**See Also** WMDocument.NewConstraint (method), WMBody (object), WMPoint (object), WMOBJECT (object), WMCell (object)

---

## WMConstraint.ActiveWhen (property)

---

**Syntax** WMConstraint.ActiveWhen

**Description** A WMCell that specifies the condition for which the constraint is active (i.e., exerts necessary force/torque to maintain the constraint).

**Comments** The ActiveWhen property itself is read-only, but you can modify the Value and Formula properties of this WMCell object (see description of the WMCell object). The condition is overridden if AlwaysActive is True, in which case the constraint is always active.

The ActiveWhen property is applicable to all WMConstraint objects.

---

**Note:** Since the property AlwaysActive is True by default, you must first set the property to False before modifying the ActiveWhen property; otherwise, Working Model will report a run-time error.

---

**Example** Sub Main()

```
' Sets the first constraint in the document's active condition
Dim Constr1 as WMConstraint
Set Constr1 = WM.ActiveDocument.Constraints.Item(1)
if Constr1 is not Nothing then
```

```
' Required before modifying ActiveWhen.
Constr1.AlwaysActive = False
Constr1.ActiveWhen.Formula = "and(frame()<50, time<2.5)"
else
 MsgBox "No constraint exists in the document"
end if
End Sub
```

**See Also** WMConstraint (object), WMConstraint.AlwaysActive (property)

## WMConstraint.ActuatorType (property)

---

**Syntax** WMConstraint.ActuatorType

**Description** A String object to specify the type of an actuator constraint.

**Comments** ActuatorType can be set to Force, Acceleration, Velocity, or Length. Applicable only when Kind is Actuator.

**Example**

```
Sub Main()
 ' Sets the driver type to sinusoidal, velocity driver
 Dim D as WMDocument : Set D = WM.Activedocument
 Dim driver as WMConstraint
 Dim Weight as WMBody
 Set Weight = D.NewBody("circle")
 Weight.radius.Value = 0.5
 Weight.PY.value = -1.5
 Set driver = D.NewConstraint("actuator")
 Set driver.Point(2).Body = Weight
 driver.ActuatorType = "Length"
 driver.Field.Formula = str$(Driver.CurrentLength)+"sin(t*5)"
 D.Run 100
End Sub
```

**See Also** WMConstraint (object)

## WMConstraint.AddVertex (method)

---

**Syntax** WMConstraint.AddVertex *index*, *x*, *y*

**Description** A method to add a new control point to the curved slot.

**Comments** AddVertex is a method which, given the index of the curved slot control point (specified in the Integer parameter *index*), adds the new control point (x, y) (Double parameters) *at* the index counter *index*. VertexCount will be updated, and the remaining control points will be shifted accordingly (i.e. their index will increment by one).

The (x, y) coordinates of the control points are measured with respect to the FOR of the curved slot (can be accessed using the Point method). Currently, WM Basic does not provide a direct way to enter polar coordinates for curved slots.

To access the existing control points, use GetVertex method.

Applicable only when Kind is CurvedSlot.

**Example**

```
Sub Main()
 ' Creates a closed curved slot and creates a polygon that has
 the
 ' save vertices
 Dim Slot as WMConstraint
 Dim Poly as WMBody
 Dim x as Double, y as Double
 Set Slot = WM.ActiveDocument.NewConstraint("curvedslot")
 ' By default, slot has three vertices (0, 0), (1,1), and (2,2)
 Slot.AddVertex 3, 2.0, 0.0
 Slot.DeleteVertex 4
 Slot.ClosedSlot = True
 ' Now the slot has (0,0), (1,1), (2,0) in that order
 Set Poly = WM.ActiveDocument.NewBody("polygon")
 For I = 1 to Slot.VertexCount
 Slot.GetVertex I, x, y
 Poly.AddVertex 1, x, y
 Next
 Poly.DeleteVertex 6 ' eliminates the default vertices
 Poly.DeleteVertex 5
 Poly.DeleteVertex 4
End Sub
```

**See Also** WMConstraint (object), WMConstraint.VertexCount (property),  
WMConstraint.GetVertex (method), WMConstraint.DeleteVertex

(method)

**WMConstraint.AppendPoint (method)****Syntax** WMConstraint.AppendPoint x,y**Description** A method to add a new point to a pulley.**Comments** AppendPoint is a method which appends a new point (x, y) (Double parameters) to a pulley. The point is added after the last point. The PointCount property will be incremented accordingly.

To access the coordinates of an existing pulley point, use the WMConstraint.Point method.

The current version of WM Basic does not allow you to delete points from the pulley. You would need to delete the entire pulley system first.

Applicable only when Kind is Pulley.

**Example** Sub Main()

```

 ' Adds a new node (2.5, 3.5) to the pulley system.
 Dim Pulley as WMConstraint
 Set Pulley = WM.ActiveDocument.Constraints.Item(1)
 if Pulley is not Nothing Then
 If Pulley.Kind = "pulley" Then
 Pulley.AppendPoint 2.5, 3.5
 MsgBox "The Pulley has
"+str$(Pulley.PointCount)+"points"
 End If
 End If
End Sub

```

**See Also** WMConstraint (object), WMConstraint.PointCount (property), WMConstraint.Point (method)**WMConstraint.AlwaysActive (property)****Syntax** WMConstraint.AlwaysActive**Description** A Boolean to indicate whether the constraint is always active.**Comments** The default value is True, meaning that the constraint is always active. If set True, AlwaysActive overrides the condition set in ActiveWhen, and you cannot modify the ActiveWhen property.

The AlwaysActive property is applicable to all WMConstraint objects.

**Example** (See `WMConstraint.ActiveWhen`)

**See Also** `WMConstraint` (object), `WMConstraint.ActiveWhen` (property)

---

### ***WMConstraint.AutoComputeGearRatio* (property)**

---

**Syntax** `WMConstraint.AutoComputeGearRatio`

**Description** A Boolean object to specify whether the gear ratio is to be automatically computed.

**Comments** When `True`, the gear ratio is automatically computed based on the relative radii of the two bodies specified as gears. If one of the bodies is not a circle and if `AutoComputeGearRatio` is `True`, the gear ratio is computed as 1.0.

When `True`, the value specified in `GearRatio` is ignored.

The default value is `True`. Applicable only when `Kind` is `Gear`.

**Example** (See `WMConstraint.GearRatio`)

**See Also** `WMConstraint` (object), `WMConstraint.GearRatio` (property)

---

### ***WMConstraint.ClosedSlot* (property)**

---

**Syntax** `WMConstraint.ClosedSlot`

**Description** A Boolean to indicate whether the curved slot geometry is open or closed.

**Comments** When `ClosedSlot` is `True`, the curved slot geometry is closed; otherwise, the geometry is open. The default value is `False`.

Applicable only when `Kind` is `CurvedSlot`.

**Example** (See `WMConstraint.AddVertex`)

**See Also** `WMConstraint` (object)

---

### ***WMConstraint.CurrentLength* (property)**

---

**Syntax** `WMConstraint.CurrentLength`

**Description** A `Double` containing the current length of a linear constraint.

**Comments** The unit is based on the current unit system. The property is read-only.

Only applicable when `Kind` is `Spring`, `SpringDamper`, `Rope`, `Separator`, and `Rod`.

**Example** (See `WMConstraint.Length`)

**See Also** `WMConstraint` (object)



---

**WMConstraint.CurrentRotation (property)**

---

- Syntax** WMConstraint.CurrentRotation
- Description** A Double containing the current rotation of a rotational constraint.
- Comments** The unit is based on the current unit system. The property is read-only.  
Only applicable when Kind is Rspring.
- Example** (See WMConstraint.Rotation)
- See Also** WMConstraint (object), WMConstraint.Rotation (property)

---

**WMConstraint.DamperK (property)**

---

- Syntax** WMConstraint.DamperK
- Description** A WMCCell object to specify the multiplicative constant for a damper component of a SpringDamper.
- Comments** The damper force is defined as  $-Kv$ . Only applicable when Kind is SpringDamper. For a Damper constraint, use the property WMConstraint.K instead.
- Example**
- ```
Sub Main()
    ' Sets the strut and simulate mass-damper-spring model
    Dim D as WMDocument : Set D = WM.Activedocument
    Dim strut as WMConstraint
    Dim Weight as WMBody
    Set Weight = D.NewBody("circle")
    Weight.radius.Value = 0.5
    Weight.PY.value = -1.5
    Set strut = D.NewConstraint("springdamper")
    Set strut.Point(2).Body = Weight
    strut.DamperK.value = 1.2 ' damping constant
    Strut.Length.Value = Strut.CurrentLength
    strut.K.Value = 25.0      ' spring constant
    D.Run 100
End Sub
```
- See Also** WMConstraint (object), WMConstraint.Exponent (property), WMConstraint.K (property)

***WMConstraint.DeleteVertex* (method)**

- Syntax** `WMConstraint.DeleteVertex index`
- Description** A method to delete a control point from a curved slot.
- Comments** `DeleteVertex` is a method which, given the index of the curved slot control point (specified in the `Integer` parameter *index*), deletes the control point. `VertexCount` will be updated, and the remaining control points will be shifted accordingly (i.e. their index will be decremented by one).
- Applicable only when `Kind` is `CurvedSlot`.
- Example** (See `WMConstraint.AddVertex`)
- See Also** `WMConstraint` (object), `WMConstraint.VertexCount` (property), `WMConstraint.GetVertex` (method), `WMConstraint.AddVertex` (method)

***WMConstraint.Elasticity* (property)**

- Syntax** `WMConstraint.Elasticity`
- Description** A `WMCell` object to specify the elasticity of a linear constraint.
- Comments** Applicable only when `Kind` is `Rope` or `Separator`.
- Example**
- ```
Sub Main()
 ' Set the elasticity of a separator (assume
 ' one exists) 1.0.

 Dim Doc as WMDocument : Set Doc = WM.ActiveDocument
 Dim Separator as WMConstraint
 Set Separator = Doc.Constraint("separator")
 If Separator is not Nothing then
 Separator.Elasticity.Value = 1.0
 Else
 MsgBox "No constraint with the name 'separator'"
 End If
End Sub
```
- See Also** `WMConstraint` (object)

---

## ***WMConstraint.Exponent* (property)**

---

- Syntax** `WMConstraint.Exponent`

- Description** An Integer to specify the exponent for displacement (for springs) or velocity (for dampers).
- Comments** The constraint force is defined by  $-Kv^e$  (for dampers) or  $Kx^e$  (for springs), where  $v$  is relative velocity,  $x$  is displacement, and  $e$  is the Exponent.  
Only applicable when Kind is Spring, Damper, Rdamper, and Rspring.
- Example** (See WMConstraint.K)
- See Also** WMConstraint, WMConstraint.K

## WMConstraint.Field (property)

- Syntax** WMConstraint.Field
- Description** A WMCell object to specify the magnitude of the motor or actuator.
- Comments** Motors and actuators are versatile constraints which can control various properties such as length, rotation, velocity, and acceleration. You must set the MotorType or ActuatorType to specify the controlled quantity, and then specify its magnitude by setting the Field property of the WMConstraint object.  
Only applicable when Kind is Motor or Actuator.
- Example** (See WMConstraint.ActuatorType and WMConstraint.MotorType)
- See Also** WMConstraint (object), WMConstraint.MotorType (property), WMConstraint.ActuatorType (property)

## WMConstraint.FR, FTheta (properties)

- Syntax** WMConstraint.FR, WMConstraint.FTheta
- Description** WMCell objects to specify the radial (FR) and angular (FTheta) components of a force constraint.
- Comments** The values are specified in the current force unit with respect to the global coordinate system.
- 
- Note:** To access FR and FTheta properties, the force magnitude specifications must be set to polar coordinates. When a force is created, the default mode is Cartesian coordinates, in which case FR and FTheta are set to Nothing. You must explicitly specify the polar mode in the Properties window for the force.
- 
- Applicable only when Kind is Force.
- Example** Sub Main()  
    ' If the first constraint in the document is a

```
' force, and if the force spec is set to Polar
' coordinates, modify (r, theta) to (10, 45).
Dim Force as WMConstraint
Set Force = WM.ActiveDocument.Constraints.Item(1)
WM.ActiveDocument.UnitSystem = "si degrees"
' Make sure that the object is Force and that the coordinate
' spec mode is Polar (otherwise FR and FTheta returns Nothing).
If (Force is not Nothing) and (Force.Kind = "force") then
 If Force.FR is not Nothing then
 Force.FTheta.Value = 45' degrees
 Force.FR.Value = 10 ' Newton
 End If
End If
End Sub
```

**See Also** WMConstraint (object), WMConstraint.FX, FY (properties)

---

## WMConstraint.FX, FY (properties)

---

**Syntax** WMConstraint.FX, WMConstraint.FY

**Description** WMCell objects to specify the X- and Y-components of a force constraint.

**Comments** The values are specified in the current force unit with respect to the global coordinate system.

Applicable only when Kind is Force.

**Example** Sub Main()

```
' Apply force at a corner of a square body.
' Let the direction of the force stay fixed with
' respect to the orientation of the body.
Dim D as WMDocument : Set D = WM.Activedocument
D.Gravity = "none"
Dim Force as WMConstraint
Dim Weight as WMBody
Set Weight = D.NewBody("square")
Weight.width.Value = 1.0
Set Force = D.NewConstraint("Force")
```

```

Set Force.Point(1).Body = Weight
Force.Point(1).PX.Value = -0.5
Force.Point(1).PY.Value = -0.5
Force.RotateWithBody = True
Force.FX.Value = 10
Force.FY.Value = 0
D.Run 50
End Sub

```

**See Also** WMConstraint (object), WMConstraint.FR, FTheta (properties)

## WMConstraint.GearRatio (property)

---

**Syntax** WMConstraint.GearRatio

**Description** A WMCell object to specify the gear ratio of a gear constraint.

**Comments** The value is ignored if AutoComputeGearRatio is True. Please note that the gear ratio is a unitless quantity.

Applicable only when Kind is Gear.

**Example** Sub Main()

```

Dim D as WMDocument : Set D=WM.ActiveDocument
Dim DriveGear as WMBody
Dim DrivenGear as WMBody
Dim GearSys as WMConstraint
Set DriveGear = D.NewBody("circle")
Set DrivenGear = D.NewBody("circle")
DriveGear.Radius.value = 1.5
DrivenGear.Radius.value = 0.5
DrivenGear.PX.Value = 3.0
Set GearSys = D.NewConstraint("gear")
GearSys.Point(1).Body = DriveGear
GearSys.Point(2).Body = DrivenGear
GearSys.AutoComputeGearRatio = False ' required
GearSys.GearRatio.Value = 5.5
' Make the rod active only while t < 2.5
GearSys.RodAlwaysActive = False

```

```
GearSys.RodActive.Formula = "time < 2.5"

' Note: At this point, circles remain unattached to

' the background; you need to attach them with

' pin joint or motor to make the gears work.

End Sub
```

**See Also** WMConstraint (object), WMConstraint.AutoComputeGearRatio (property)

---

## WMConstraint.GetVertex (method)

---

**Syntax** WMConstraint.GetVertex *index*, *x*, *y*

**Description** A method to obtain the x- and y-coordinates of a specified control point of a curved slot.

**Comments** GetVertex is a method which, given the index of a curved slot control point (specified in the Integer parameter *index*), fills the Double parameters *x*, *y* as the coordinates of the specified control point.  
Applicable only when Kind is CurvedSlot.

**Example** (See WMConstraint.AddVertex)

**See Also** WMConstraint (object), WMConstraint.VertexCount (property), WMConstraint.AddVertex (method), WMConstraint.DeleteVertex (method)

---

## WMConstraint.Internal (property)

---

**Syntax** WMConstraint.Internal

**Description** A Boolean to specify whether the gear system works as an internal-external gear pair.

**Comments** When Internal is True, one of the bodies involved in the gear constraint (specified in InternalBody) becomes the internal gear. Use InternalBody to specify which gear is acting as an internal gear.  
Default value is False.  
Applicable only when Kind is Gear.

**Example** (See WMConstraint.GearRatio)

**See Also** WMConstraint (object), WMConstraint.InternalBody (property)

---

## WMConstraint.InternalBody (property)

---

**Syntax** WMConstraint.InternalBody

**Description** An Integer to specify the body acting as the internal gear. Applicable only when Kind is Gear, and only if Internal is True.

**Comments** The Integer property returns 1 or 2. The number corresponds to which body was selected first when the gear was created.

**Example**

```
Sub Main()
 ' Provided that an internal gear constraint exists in doc,
 ' reports which body is acting as the internal gear.

 Dim Doc as WMDocument
 Dim Gear as WMConstraint
 Set Doc = WM.ActiveDocument
 Set Gear = Doc.Constraint("gear")

 If Gear is not Nothing then
 If Gear.Internal then
 MsgBox "Internal gear is "+str$(Gear.InternalBody)
 Else
 MsgBox "Gear found, but it is not an internal gear"
 End If
 End If
End Sub
```

**See Also** WMConstraint (object), WMConstraint.Internal (property)

## WMConstraint.K (property)

**Syntax** WMConstraint.K

**Description** A WMCell object to specify the multiplicative constant for the constraint force.

**Comments** The constraint force is defined by  $-Kv^e$  (for dampers) or  $-Kx^e$  (for springs), where  $v$  is relative velocity,  $x$  is displacement, and  $e$  is the exponent.

Only applicable when Kind is Spring, Damper, Rdamper, SpringDamper and Rspring.

**Example**

```
Sub Main()
 ' Simulate mass-spring model; spring is -kx^2

 Dim D as WMDocument : Set D = WM.Activedocument
 Dim Spring as WMConstraint
 Dim Weight as WMBody
```

```
Set Weight = D.NewBody("circle")
Weight.radius.Value = 0.5
Weight.PY.value = -1.5

Set Spring = D.NewConstraint("spring")
Set Spring.Point(2).Body = Weight
Spring.Exponent = 2
Spring.K.Value = 25.0 ' spring constant
Spring.Length.Value = Spring.CurrentLength

D.Run 100

End Sub
```

**See Also** WMConstraint, WMConstraint.Exponent

---

## WMConstraint.Kind (property)

---

**Syntax** WMConstraint.Kind

**Description** A String to indicate the type of a constraint.

**Comments** The Kind property is read-only and automatically defined by Working Model when the constraint is created. See the section on WMConstraint (object) for the range of values accepted as this String property.

The Kind property is applicable to all WMConstraint objects.

**Example** (See WMConstraint.ActuatorType)

**See Also** WMConstraint (object)

---

## WMConstraint.Length (property)

---

**Syntax** WMConstraint.Length

**Description** A WMCell object to specify rest length of a linear constraint.

**Comments** The unit is based on the current unit system.

Only applicable when Kind is Spring, SpringDamper, Rope, Separator, and Rod.

**Example**

```
Sub Main()
 ' Sets a spring-mass model, where the rest length
 ' of the spring is half of the initial length.
 ' (i.e. spring is under tension initially)

 Dim D as WMDocument : Set D = WM.ActiveDocument
```



```

Dim Spring as WMConstraint
Dim Pendulum as WMBody
Set Pendulum = D.NewBody("circle")
Pendulum.Radius.Value = 0.5
Pendulum.PY.Value = -2.0
Set Spring = D.NewConstraint("spring")
SetSpring.Point(1).Body = Pendulum
Spring.Length.Value = Spring.CurrentLength / 2
D.Run 45
D.Reset
End Sub

```

**See Also** WMConstraint (object)

## WMConstraint.MotorType (property)

---

**Syntax** WMConstraint.MotorType

**Description** A String object to specify the type of a motor constraint.

**Comments** MotorType can be set to Torque, Acceleration, Velocity, Or Rotation. Applicable only when Kind is Motor.

**Example**

```

Sub Main()
 ' Sets the motor type to sinusoidal, velocity driver
 Dim D as WMDocument : Set D = WM.Activedocument
 Dim Motor as WMConstraint
 Dim Spinner as WMBody
 Set Spinner = D.NewBody("square")
 Spinner.Width.Value = 1.0
 Set Motor = D.NewConstraint("motor")
 Set Motor.Point(2).Body = Spinner
 Motor.MotorType = "Velocity"
 Motor.Field.Formula = "200*sin(t*8)"
 D.Run 100
End Sub

```

**See Also** WMConstraint (object)

## WMConstraint.Point (method)

**Syntax** WMConstraint.Point(*index*)

**Description** Returns a WMPoint object that represents the point element of a linear constraint.

**Comments** The parameter *index* is an Integer that distinguish one point from another. The value of *index* can be either 1 or 2 for all linear constraints except for a pulley. For a pulley system, *index* can range from 1 to the number of points embedded in the pulley. Refer to the WMConstraint.PointCount property for information.

The Point method returns a WMPoint object. Refer to the section on WMPoint for more information .

Applicable only when Kind is Spring, Damper, SpringDamper, Rod, Separator, Rope, Actuator, Pulley, Pin, SquarePin, Rspring, Rdamper, Motor, Hslot, Vslot, KeyedHslot, KeyedVslot, and CurvedSlot.

To append a point to a pulley system, use the AppendPoint method of the WMConstraint object.

The Point method returns the following depending on the Kind of the WMConstraint object.

| Kind                                                         | Point (1)                                                                             | Point (2)                                                                              |
|--------------------------------------------------------------|---------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Spring, Damper, SpringDamper, Rod, Separator, Rope, Actuator | (If the constraint was created interactively with the mouse) the first point created. | (If the constraint was created interactively with the mouse) the second point created. |
| Pulley                                                       | The first node created.                                                               | The second node created.                                                               |
| Pin, SquarePin, Rspring, Rdamper, Motor                      | Point attached to the body on the lower layer.                                        | Point attached to the body on the upper layer.                                         |
| Hslot, Vlost, KeyedHSlot, KeyedVSlot, CurvedSlot             | Point representing the slot element, or the FOR of the slot .                         | Point attached to the body moving along the slot.                                      |

**Note:** When you create a pin joint or locked joint by performing JOIN, the body created *earlier* belongs to the lower layer.

**Example** Sub Main()

```
' Create two links and connect them with a pin joint.
```

```
Dim D as WMDocument : Set D = WM.ActiveDocument
Dim Joint as WMConstraint
Dim Link1 as WMBody
Dim Link2 as WMBody
' Create two bodies
Set Link1 = D.NewBody("rectangle")
Link1.Width.Value = 2.0 : Link1.Height.Value = 0.5
Set Link2 = D.NewBody("rectangle")
Link2.Width.Value = 0.5 : Link2.Height.Value = 2.0
' Create a joint and attach it to the links
Set Joint = D.NewConstraint("pin")
SetJoint.Point(2).Body = Link1
Joint.Point(2).PX.Value = -0.75
Joint.Point(2).PY.Value = 0
Set Joint.Point(1).Body = Link2
Joint.Point(1).PX.Value = 0
Joint.Point(1).PY.Value = 0.75
' Modify the position of the linkage
Link2.PX.Value = 0.0 : Link2.PY.Value = 1.0
End Sub
```

**See Also** WMConstraint (object), WMConstraint.PointCount (property),  
WMConstraint.AppendPoint (method), WMPoint (object)

---

## **WMConstraint.PointCount (property)**

---

**Syntax** WMConstraint.PointCount

**Description** An Integer which shows the number of point elements embedded in a constraint (endpoints for most constraints, or nodes for pulleys).

**Comments** Read-only. Most useful when kind is Pulley. For rotational/linear constraints and joints, PointCount returns two. For force, torque, and a slot element, PointCount returns 1. To append a point to the pulley, use the WMConstraint.AppendPoint method.

**Example** (See WMConstraint.AppendPoint)

**See Also** WMConstraint (object), WMConstraint.AppendPoint (method),  
WMConstraint.Point (method)

---

**WMConstraint.RodActive (property)**


---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMConstraint.RodActive</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Description</b> | A <code>WMCell</code> object to specify the condition under which the built-in rod constraint in the gear becomes an active constraint.                                                                                                                                                                                                                                                                                                                                                               |
| <b>Comments</b>    | The value specified in <code>RodActive</code> is overridden if <code>RodAlwaysActive</code> is <code>True</code> .<br><br><b>Note:</b> Since the property <code>RodAlwaysActive</code> is <code>True</code> by default, you must first set the property to <code>False</code> before modifying the <code>RodActive</code> property; otherwise, Working Model will not allow modification of the <code>RodActive</code> property.<br><br>Applicable only when <code>Kind</code> is <code>Gear</code> . |
| <b>Example</b>     | (See <code>WMConstraint.GearRatio</code> )                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>See Also</b>    | <code>WMConstraint</code> (object), <code>WMConstraint.RodAlwaysActive</code> (property), <code>WMConstraint.GearRatio</code> (property)                                                                                                                                                                                                                                                                                                                                                              |

---

**WMConstraint.RodAlwaysActive (property)**


---

|                    |                                                                                                                                                                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMConstraint.RodAlwaysActive</code>                                                                                                                                                                                                       |
| <b>Description</b> | A <code>Boolean</code> to specify whether the build-in rod constraint in the gear should always be active.                                                                                                                                      |
| <b>Comments</b>    | If set <code>True</code> , <code>RodAlwaysActive</code> overrides conditions that may have been given in <code>RodActive</code> . The default value is <code>True</code> .<br><br>Applicable only when <code>Kind</code> is <code>Gear</code> . |
| <b>Example</b>     | (See <code>WMConstraint.GearRatio</code> )                                                                                                                                                                                                      |
| <b>See Also</b>    | <code>WMConstraint</code> (object), <code>WMConstraint.RodActive</code> (property), <code>WMConstraint.GearRatio</code> (property)                                                                                                              |

---

**WMConstraint.RotateWithBody (properties)**


---

|                    |                                                                                                                                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMConstraint.RotateWithBody</code>                                                                                                                                                                                                                                                           |
| <b>Description</b> | A <code>Boolean</code> object to specify whether the direction of the force is to be fixed with respect to the body to which it is attached.                                                                                                                                                       |
| <b>Comments</b>    | When <code>RotateWithBody</code> is <code>True</code> , the direction of the force is fixed with respect to the body (i.e., retains the direction specified at the initial frame). The default value is <code>False</code> .<br><br>Applicable only when <code>Kind</code> is <code>Force</code> . |
| <b>Example</b>     | (See <code>WMConstraint.FX</code> , <code>FY</code> )                                                                                                                                                                                                                                              |

**See Also** WMConstraint (object), WMConstraint.FX, FY (properties)

## WMConstraint.Rotation (property)

**Syntax** WMConstraint.Rotation

**Description** A WMCell object that specifies the rest rotation of a rotational constraint.

**Comments** The unit for rotation is based on the current unit system.  
Only applicable when Kind is RSpring.

**Example**

```
Sub Main()
 ' set up a torsion pendulum
 Dim D as WMDocument : Set D = WM.ActiveDocument
 D.Gravity = "none"
 Dim Spring as WMConstraint
 Dim Pendulum as WMBody
 Set Pendulum = D.NewBody("rectangle")
 Pendulum.Width.Value = 0.5
 Pendulum.Height.Value = 2.0
 Set Spring = D.NewConstraint("Rspring")
 SetSpring.Point(2).Body = Pendulum
 Spring.Point(2).PY.Value = 0.75
 Pendulum.PR.Value = 45
 Spring.Rotation.Value = Spring.CurrentRotation * 2
 D.Run 45
 D.Reset
End Sub
```

**See Also** WMConstraint (object)

## WMConstraint.Torque (property)

**Syntax** WMConstraint.Torque

**Description** A WMCell object to specify the magnitude of Torque.

**Comments** The magnitude is interpreted in the current torque unit. You can also use the Field property to specify the torque (see Example below).  
Applicable only when Kind is Torque.

**Example**

```

Sub Main()
 ' Creates a body and applies torque to it.
 Dim Doc as WMDocument : Set Doc = WM.ActiveDocument
 Doc.Gravity = "none"
 Dim Body as WMBody : Set Body = Doc.NewBody("square")
 Body.Mass.Value = 5
 Dim T as WMConstraint
 Set T = Doc.NewConstraint("torque")
 Set T.Point(1).Body = Body
 T.Torque.Value = 10 ' T.Force.Value = 10 would work, too
 Doc.Run 50
End Sub

```

**See Also** WMConstraint (object), WMConstraint.Field (property)

## WMConstraint.VertexCount (property)

---

**Syntax** WMConstraint.VertexCount

**Description** An Integer which returns the number of control points embedded in a curved slot joint.

**Comments** Read-only. Applicable only when Kind is CurvedSlot.

**Example** (See WMConstraint.AddVertex)

**See Also** WMConstraint (object), WMConstraint.GetVertex (method), WMConstraint.AddVertex (method), WMConstraint.DeleteVertex (method)

## WMDocument (object)

---

**Syntax** WMDocument

**Description** An object which provides an interface to Working Model document.

**Comments** Most of Working Model Basic functionalities operate on a Working Model document. You usually declare a WMDocument object before running a simulation or manipulating individual Working Model objects, such as constraints and bodies.

WMDocument has the following properties and methods. Please refer to individual sections for complete descriptions.

### Property/MethodDescription

---

|                                    |                                                                                                |
|------------------------------------|------------------------------------------------------------------------------------------------|
| <code>AirResistanceType</code>     | Specifies the type of air resistance.                                                          |
| <code>AirResistanceV2Coeff</code>  | Specifies the high air resistance coefficient.                                                 |
| <code>AirResistanceVCoeff</code>   | Specifies the standard air resistance coefficient.                                             |
| <code>AnimationStep</code>         | Specifies the Animation Step of the simulation.                                                |
| <code>AssemblyError</code>         | Specifies the Assembly Error.                                                                  |
| <code>AutoAnimationStep</code>     | Enables or disables the automatic Animation Step determination.                                |
| <code>AutoAssemblyError</code>     | Enables or disables the automatic Assembly Error determination.                                |
| <code>AutoEraseTrack</code>        | Specifies whether the AutoErase Track feature is active.                                       |
| <code>AutoIntegratorError</code>   | Enables or disables the automatic Integrator Error determination.                              |
| <code>AutoOverlapError</code>      | Enables or disables the automatic Overlap Error determination.                                 |
| <code>AutoSignificantDigits</code> | Enables or disables the automatic Significant Digits determination.                            |
| <code>Bodies</code>                | The collection of all <code>WMBody</code> objects in the document.                             |
| <code>Body</code>                  | Returns the specified <code>WMBody</code> object.                                              |
| <code>ChargeUnit</code>            | Specifies the charge unit.                                                                     |
| <code>Close</code>                 | Closes the document.                                                                           |
| <code>CombineTapeScroll</code>     | Specifies whether the tape control display is combined in line with the horizontal scroll bar. |
| <code>Constraint</code>            | Returns the specified <code>WMConstraint</code> object.                                        |
| <code>Constraints</code>           | The collection of all <code>WMConstraint</code> objects in the document.                       |
| <code>ControlsLocked</code>        | Lock Control.                                                                                  |
| <code>Copy</code>                  | Copies selected set of objects to the Clipboard.                                               |
| <code>CurrentFrame</code>          | Shows the current frame number of the simulation.                                              |
| <code>Cut</code>                   | Copies selected set of objects to the Clipboard and deletes the objects from the document.     |
| <code>DecimalDigits</code>         | Specifies the number of decimal digits displayed in meters and dialogs.                        |
| <code>DecimalFormat</code>         | Specifies the display format for floating point numbers.                                       |
| <code>Delete</code>                | Permanently deletes selected set of objects.                                                   |
| <code>DistanceUnit</code>          | Specifies the distance unit.                                                                   |
| <code>ElectricPotentialUnit</code> |                                                                                                |

|                                                                                  |                                                                                            |
|----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
|                                                                                  | Specifies the electric potential unit.                                                     |
| <code>ElectrostaticConst</code>                                                  | Specifies the electrostatic constant factor.                                               |
| <code>ElectrostaticsOn</code>                                                    | Specifies whether the electrostatic forces are active between bodies.                      |
| <code>EnergyUnit</code>                                                          | Specifies the energy unit.                                                                 |
| <code>EraseTrack</code>                                                          | Erases traced simulation tracks left on the document window.                               |
| <code>ExportDXF</code>                                                           | Exports a DXF file.                                                                        |
| <code>ExportMeterData</code>                                                     | Exports meter data to a file.                                                              |
| <code>ExportStartFrame</code>                                                    | Specifies the initial frame number when exporting data to a file.                          |
| <code>ExportStopFrame</code>                                                     | Specifies the final frame number when exporting data to a file.                            |
| <code>ForceFieldFX</code> , <code>ForceFieldFY</code> , <code>ForceFieldT</code> | Specifies the force field components.                                                      |
| <code>ForceFieldType</code>                                                      | Specifies the force field type.                                                            |
| <code>ForceUnit</code>                                                           | Specifies the force unit.                                                                  |
| <code>FrequencyUnit</code>                                                       | Specifies the frequency unit.                                                              |
| <code>Gravity</code>                                                             | Specifies the type of gravitational force acting in the document.                          |
| <code>HistoryFrames</code>                                                       | Shows the number of frames currently stored in memory.                                     |
| <code>ImportDXF</code>                                                           | Imports a DXF file.                                                                        |
| <code>Input</code>                                                               | Returns the specified <code>WMInput</code> object.                                         |
| <code>Inputs</code>                                                              | Collection of all <code>WMInput</code> objects in the document.                            |
| <code>Integrator</code>                                                          | Specifies the integration method.                                                          |
| <code>IntegratorError</code>                                                     | Specifies the Integrator Error.                                                            |
| <code>Join</code>                                                                | Joins objects.                                                                             |
| <code>LinearGravityConst</code>                                                  | Specifies the linear gravitational constant.                                               |
| <code>MassUnit</code>                                                            | Specifies the mass unit.                                                                   |
| <code>Name</code>                                                                | The document name (file name).                                                             |
| <code>NewBody</code>                                                             | Creates a new <code>WMBody</code> object.                                                  |
| <code>NewConstraint</code>                                                       | Creates a new <code>WMConstraint</code> object.                                            |
| <code>NewInput</code>                                                            | Creates a new <code>WMInput</code> object.                                                 |
| <code>NewOutput</code>                                                           | Creates a new <code>WMOutput</code> object.                                                |
| <code>NewPoint</code>                                                            | Creates a new <code>WMPoint</code> object.                                                 |
| <code>Object</code>                                                              | Returns the specified <code>WMObject</code> object (regardless of its <code>Kind</code> ). |



|                        |                                                                                 |
|------------------------|---------------------------------------------------------------------------------|
| Objects                | Collection of all <code>WMObject</code> objects in the document.                |
| Output                 | Returns the specified <code>WMOutput</code> object.                             |
| Outputs                | Collection of all <code>WMOutput</code> objects in the document.                |
| OverlapError           | Specifies the Overlap Error.                                                    |
| Paste                  | Pastes objects from Clipboard to the Working Model document.                    |
| PlanetaryGravityConst  | Specifies the planetary gravitational constant.                                 |
| PlayerMode             | Specifies whether the document is in Player mode.                               |
| Point                  | Returns the specified <code>WMPoint</code> object.                              |
| Points                 | Collection of <code>WMPoint</code> objects in the document.                     |
| PowerUnit              | Specifies the power unit.                                                       |
| Reset                  | Resets the Working Model simulation.                                            |
| RotationalVelocityUnit | Specifies the rotational velocity unit.                                         |
| RotationUnit           | Specifies the rotation unit.                                                    |
| Run                    | Runs the Working Model simulation.                                              |
| Save                   | Saves the Working Model simulation under the current filename.                  |
| SaveAs                 | Saves the Working Model simulation under the specified name.                    |
| ScaleFactor            | Specifies the scale factor of the Working Model document screen.                |
| ScrollTo               | Scrolls the document screen to a specific location.                             |
| Select                 | Selects or de-selects Working Model objects in the document.                    |
| SelectAll              | Selects or de-selects all Working Model objects simultaneously in the document. |
| Selection              | Contains the objects currently selected in the document.                        |
| ShowCoordinates        | Show or hide the Coordinates bar.                                               |
| ShowGridLines          | Show or hide the grid lines.                                                    |
| ShowHelpRibbon         | Show or hide the Help Ribbon.                                                   |
| ShowRulers             | Show or hide the rulers.                                                        |
| ShowScrollBars         | Show or hide the scroll bars.                                                   |
| ShowTapeControl        | Show or hide the tape control.                                                  |
| ShowToolPalette        | Show or hide the Toolbar.                                                       |

## 558 Working Model Basic User's Manual

---

|                         |                                                                                 |
|-------------------------|---------------------------------------------------------------------------------|
| ShowXYAxes              | Show or hide the x- and y-axes.                                                 |
| SimulationMode          | Sets the simulation mode (e.g. Fast, Accurate)                                  |
| SignificantDigits       | Specifies the number of significant digits.                                     |
| SkipFrames              | Specifies the frame skip rate.                                                  |
| Split                   | Splits the selected bodies or constraints.                                      |
| StartHere               | Resets the simulation history.                                                  |
| TimeUnit                | Specifies the time unit.                                                        |
| Tracking                | Specifies the tracking rate.                                                    |
| UnitSystem              | Specifies the current unit system.                                              |
| Update                  | Updates the Working Model document screen to reflect changes in WMCell objects. |
| VariableIntegrationStep | Sets the integration step to be fixed or variable.                              |
| VelocityUnit            | Specifies the (linear) velocity unit.                                           |
| ViewWidth               | Specifies the view width (scaled length of the horizontal ruler).               |
| WarnInconsistent        | Enables or disables the inconsistent constraint warning.                        |
| WarnLargeVorA           | Enables or disables the large velocity/acceleration warning.                    |
| WarnOverlap             | Enables or disables the initial body overlap warning.                           |
| WarnRedundant           | Enables or disables the redundant constraint warning.                           |

**Example**

```
Sub Main()
 Dim WM1 as WMDocument
 Set WM1 = WM.ActiveDocument
 MsgBox "The currently active document is " + WM1.Name
End Sub
```

**See Also** WM (constant)

## WMDocument.AirResistanceType (property)

---

|                    |                                                                                          |
|--------------------|------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.AirResistanceType                                                             |
| <b>Description</b> | A String which controls the type of air resistance acting in the Working Model document. |
| <b>Comments</b>    | The AirResistanceType property can take one of the following.                            |
| <b>Value</b>       | <b>Unit Description</b>                                                                  |

---

|          |                                                                                                               |
|----------|---------------------------------------------------------------------------------------------------------------|
| none     | No air resistance.                                                                                            |
| standard | Standard air resistance, which acts on a body proportional to its cross sectional width and velocity.         |
| high     | High air resistance, which acts on a body proportional to its cross sectional width and the squared velocity. |

The default value for every new document is `none`.

Use the properties `AirResistanceVCoeff` and `AirResistanceV2Coeff` to control standard and high air resistance forces, respectively.

**Example**

```
Sub Main()
 Dim D as WMDocument : Set D = WM.ActiveDocument
 D.AirResistanceType = "Standard"
 D.AirResistanceV2Coeff = 7.0
End Sub
```

**See Also**   `WMDocument (object)`, `WMDocument.AirResistanceVCoeff(property)`, `WMDocument.AirResistanceV2Coeff(property)`

---

## ***WMDocument.AirResistanceV2Coeff (property)***

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMDocument.AirResistanceV2Coeff</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Description</b> | A <code>Double</code> which specifies the high air resistance coefficient in the Working Model document.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Comments</b>    | <p>You must turn on the high air resistance first in order for this constant to take effect in the Working Model document (see <code>WMDocument.AirResistanceType</code>). The property <code>AirResistanceV2Coeff</code> pertains to the coefficient <math>k</math> used in computing <math>F = k/v^2 A</math>, where <math>F</math> is the force acting on every body (whose cross-sectional width is <math>A</math> (projected to the direction of <math>v</math>) and the velocity is <math>v</math>) in the document.</p> <p>The default value of <code>AirResistanceV2Coeff</code> is <math>5.0 \text{ kg /m sec}^2</math> (<math>0.007 \text{ lb / in sec}^2</math>). The numerical value is automatically adjusted to accommodate the current unit system in order to retain the physical magnitude of the constant.</p> <p>See <code>WMDocument.AirResistanceType</code> to specify the high air resistance constant.</p> |
| <b>Example</b>     | <pre>Sub Main()     Dim D as WMDocument : Set D = WM.ActiveDocument     D.AirResistanceType = "High"</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

```
D.AirResistanceV2Coeff = 7.0
```

```
End Sub
```

**See Also** WMDocument (object), WMDocument.AirResistanceType(property),  
WMDocument.AirResistanceVCoeff(property)

---

## WMDocument.AirResistanceVCoeff (property)

---

**Syntax** WMDocument.AirResistanceVCoeff

**Description** A Double which specifies the standard air resistance coefficient in the Working Model document.

**Comments** You must turn on the standard air resistance first in order for this constant to take effect in the Working Model document (see WMDocument.AirResistanceType). The property AirResistanceVCoeff pertains to the coefficient  $k$  used in computing  $F = k/v/A$ , where  $F$  is the force acting on every body (whose cross-sectional width is  $A$  (projected to the direction of  $v$ ) and the velocity is  $v$ ) in the document.

The default value of AirResistanceVCoeff is 5.0 kg /m sec (0.28 lb / in sec). The numerical value is automatically adjusted to accommodate the current unit system in order to retain the physical magnitude of the constant.

See WMDocument.AirResistanceType to specify the standard air resistance constant.

**Example** (See WMDocument.AirResistanceType)

**See Also** WMDocument (object), WMDocument.AirResistanceType(property),  
WMDocument.AirResistanceV2Coeff(property)

---

## WMDocument.AnimationStep (property)

---

**Syntax** WMDocument.AnimationStep

**Description** The animation time step of the Working Model document.

**Comments** AnimationStep is a Double property. The property is exactly equivalent to the Animation Step in the Accuracy dialog of Working Model.

For a new document, the default value of AnimationStep is determined by Working Model automatically. The value varies according to the size and unit system of the model.

In order to specify AnimationStep, you must first set AutoAnimationStep to False.

**Example** Sub Main()

```
' Uses Fast simulation mode for the active document, and
' set the Animation Step to 0.01

WM.ActiveDocument.SimulationMode = "Fast"

' At this point, AutoAnimationStep is True and AnimationStep
' has the value automatically determined by Working Model.

WM.ActiveDocument.AutoAnimationStep = False

WM.ActiveDocument.AnimationStep = 0.1

End Sub
```

**See Also** WM.Document (object), WM.Document.AutoAnimationStep (property)

## WM.Document.AssemblyError (property)

---

**Syntax** WM.Document.AssemblyError

**Description** Specifies the Assembly Error of the Working Model document.

**Comments** AssemblyError is a Double property. The property is equivalent to the Assembly Error in the Accuracy dialog of Working Model.

For a new document, the default value of AssemblyError is determined by Working Model automatically. The value varies according to the size and unit system of the model.

In order to specify AssemblyError, you must first set AutoAssemblyError to False.

**Example**

```
Sub Main()

' Uses Fast simulation mode for the active document, and
' set the Assembly Error to 0.01

WM.ActiveDocument.SimulationMode = "Fast"

' At this point, AutoAssemblyError is True and
' AssemblyError has the value automatically determined by
' Working Model.

WM.ActiveDocument.AutoAssemblyError = False

WM.ActiveDocument.AssemblyError = 0.01

End Sub
```

**See Also** WM.Document (object), WM.Document.AutoAssemblyError (property)

## WM.Document.AutoAnimationStep (property)

---

**Syntax** WM.Document.AutoAnimationStep

- Description** Specifies whether `AnimationStep` for `WMDocument` object is to be determined automatically.
- Comments** `AutoAnimationStep` is a Boolean property. Setting the property to `True` is equivalent to checking the Automatic radio button under Animation Step in the Accuracy dialog of Working Model.
- When the property is `True`, Working Model automatically determines an appropriate animation time step.
- The default value of `AutoAnimationStep` is `True`.
- If you set the `AutoAnimationStep` to be `False`, the value of `AnimationStep` remains unchanged. You need to directly modify the value of `AnimationStep` if necessary.
- Example** See `WMDocument.AnimationStep`.
- See Also** `WMDocument` (object), `WMDocument.AnimationStep` (property)

---

### ***WMDocument.AutoAssemblyError* (property)**

---

- Syntax** `WMDocument.AutoAssemblyError`
- Description** Specifies whether `AssemblyError` for `WMDocument` object is to be determined automatically.
- Comments** `AutoAssemblyError` is a Boolean property. Setting the property to `True` is equivalent to checking the Automatic radio button under Assembly Error in the Accuracy dialog of Working Model. Please refer to the *Working Model User's Manual* for descriptions of Assembly Error.
- When the property is `True`, Working Model will automatically determine an appropriate size of Assembly Error.
- The default value of `AutoAssemblyError` is `True`.
- If you set the `AutoAssemblyError` to be `False`, the value of `AssemblyError` remains unchanged. You need to modify the `AssemblyError` property directly if necessary.
- Example** See `WMDocument.AssemblyError`.
- See Also** `WMDocument` (object), `WMDocument.AssemblyError` (property)

---

### ***WMDocument.AutoEraseTrack* (property)**

---

- Syntax** `WMDocument.AutoEraseTrack`
- Description** A Boolean to specify whether the AutoErase Track feature is active.
- Comments** When the AutoErase Track feature is active, simulation tracks are erased automatically whenever the document is modified. Please refer to the

*Working Model User's Manual* for more information on this feature.

AutoErase Track feature is active when `AutoEraseTrack` property is `True`.

The default value of `AutoEraseTrack` is `True`.

**Example** (See `WMDocument.EraseTrack`)

**See Also** `WMDocument.EraseTrack` (method)

---

## ***WMDocument.AutoIntegratorError (property)***

---

**Syntax** `WMDocument.AutoIntegratorError`

**Description** Specifies whether `IntegratorError` for `WMDocument` object is to be determined automatically.

**Comments** `AutoIntegratorError` is a Boolean property. Setting the property to `True` is equivalent to checking the Automatic radio button under Integrator Error in the Accuracy dialog of Working Model. Please refer to the *Working Model User's Manual* for descriptions of Integrator Error.

When the property is `True`, Working Model will automatically determine an appropriate size of Integrator Error.

The default value of `AutoIntegratorError` is `True`.

If you set the `AutoIntegratorError` to be `False`, the value of `IntegratorError` remains unchanged. You need to modify the `IntegratorError` property directly if necessary.

**Example** See `WMDocument.IntegratorError`.

**See Also** `WMDocument` (object), `WMDocument.IntegratorError` (property)

---

## ***WMDocument.AutoOverlapError (property)***

---

**Syntax** `WMDocument.AutoOverlapError`

**Description** Specifies whether `OverlapError` for `WMDocument` object is to be determined automatically.

**Comments** `AutoOverlapError` is a Boolean property. Setting the property to `True` is equivalent to checking the Automatic radio button under Overlap Error in the Accuracy dialog of Working Model. Please refer to the *Working Model User's Manual* for descriptions of Overlap Error.

When the property is `True`, Working Model will automatically determine an appropriate size of Overlap Error.

The default value of `AutoOverlapError` is `True`.

If you set the `AutoOverlapError` to be `False`, the value of

`OverlapError` remains unchanged. You need to modify the `OverlapError` property directly if necessary.

**Example** See `WMDocument.OverlapError`.

**See Also** `WMDocument` (object), `WMDocument.OverlapError` (property)

---

## ***WMDocument.AutoSignificantDigits* (property)**

---

**Syntax** `WMDocument.AutoSignificantDigits`

**Description** Specifies whether `SignificantDigits` for `WMDocument` object is to be determined automatically.

**Comments** `AutoSignificantDigits` is a Boolean property. Setting the property to `True` is equivalent to checking the Automatic radio button under Significant Digits in the Accuracy dialog of Working Model. Please refer to the *Working Model User's Manual* for descriptions of Significant Digits.

When the property is `True`, Working Model will automatically determine an appropriate size of Significant Digits.

The default value of `AutoSignificantDigits` is `True`.

If you set the `AutoSignificantDigits` to be `False`, the value of `SignificantDigits` remains unchanged. You need to modify the `SignificantDigits` property directly if necessary.

**Example** See `WMDocument.SignificantDigits`.

**See Also** `WMDocument` (object), `WMDocument.SignificantDigits` (property)

---

## ***WMDocument.Bodies* (property)**

---

**Syntax** `WMDocument.Bodies`

**Description** Returns the collection of all `WMBody` objects present in the document.

**Comments** The `Bodies` property is a collection of all `WMBody` objects present in the document. Like any other Collection object, you can use the `Item` method to access a specific body within the collection.

The `Bodies` property is read-only.

If you wish to access a single body in the document instead of a collection, use the `Body` method of the `WMDocument` object.

The index parameter given to the `Item` method is sequential within the set of bodies in the document. For example, if a document `Doc` has 10 objects (bodies, constraints, etc.) and 3 of the 10 objects are bodies, these bodies may be referred to as `Doc.Body(3)`, `Doc.Body(6)` and `Doc.Body(7)` with the `Body` method. With the `Bodies` property, these objects are referred to



as `Doc.Bodies.Item(1)`, `Doc.Bodies.Item(2)`, and `Doc.Bodies.Item(3)`, but not necessarily in that order. The `Bodies` property is provided as a convenient tool to access all bodies in a loop statement (see Example), and the indices given to the `Item` are not permanently linked to individual `WMBody` objects.

**Example**

```
Sub Main()
 ' change the name of all bodies to "truss member"

 Dim D as WMDocument
 Dim B as WMBody
 Set D = WM.ActiveDocument
 For I = 1 to D.Bodies.Count
 D.Bodies.Item(I).Name = "truss member"
 Next
End Sub
```

**See Also** `WMDocument` (object), `WMDocument.Body` (method), `WMBody` (object), `Collection` (topic)

## WMDocument.Body (method)

**Syntax** `WMDocument.Body(name | id)`

**Description** Returns the first `WMBody` object that matches the given name or ID number, or the special value *Nothing* if none is found.

**Comments** The `WMDocument.Body` method takes one or the other of the following parameters:

| Parameter   | Description                                                                                                                 |
|-------------|-----------------------------------------------------------------------------------------------------------------------------|
| <i>name</i> | A String containing the name of the <code>WMBody</code> object to be searched. The string match will not be case-sensitive. |

|           |                                                           |
|-----------|-----------------------------------------------------------|
| <i>id</i> | An Integer specifying the ID of the <code>WMBody</code> . |
|-----------|-----------------------------------------------------------|

**Note:** The `Body` method returns a single body, whereas the `Bodies` property of a `WMDocument` returns the *collection* of all bodies within the document.

**Example**

```
Sub Main()
 ' Outputs the ID number of a body named "Cam Lobe".

 Dim D as WMDocument
 Dim B as WMBody
 Set D = WM.ActiveDocument
```

```
Set B = D.Body("Cam Lobe")
If B is not Nothing then
 MsgBox "Cam Lobe ID: "+str$(B.ID)
Else
 MsgBox "No object by the name of Cam Lobe found"
End If
End Sub
```

**See Also** WMDocument (object), WMDocument.Bodies (property), WMBody (object)

---

## WMDocument.ChargeUnit (property)

---

|                    |                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.ChargeUnit                                                                                                                                                                                                                                                                                                        |
| <b>Description</b> | A String which specifies the charge unit in the Working Model document.                                                                                                                                                                                                                                                      |
| <b>Comments</b>    | The ChargeUnit property can take one of two values as follows.                                                                                                                                                                                                                                                               |
| <b>Value</b>       | <b>Unit Description</b>                                                                                                                                                                                                                                                                                                      |
| Coulombs           | Coulombs.                                                                                                                                                                                                                                                                                                                    |
| Statcoulombs       | Statcoulombs.                                                                                                                                                                                                                                                                                                                |
|                    | Both lower- and upper-case letters are accepted. The default value of the ChargeUnit property is "Coulombs".                                                                                                                                                                                                                 |
|                    | The property may be overwritten when the user explicitly specifies the UnitSystem property of the document, because each unit system has a set of specifications for all measurement units. For example, the SI unit system automatically changes distance unit to meters, mass unit to kilograms, and time unit to seconds. |
| <b>Example</b>     | (See WMDocument.UnitSystem)                                                                                                                                                                                                                                                                                                  |
| <b>See Also</b>    | WMDocument.UnitSystem (property), WMDocument.DecimalFormat (property), WMDocument.DecimalDigits (property)                                                                                                                                                                                                                   |

---

## WMDocument.Close (method)

---

|                    |                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.Close                                                                                                                               |
| <b>Description</b> | Closes the Working Model document.                                                                                                             |
| <b>Comments</b>    | If the document has been modified without being saved, Working Model will display a dialog asking whether the user wishes to save the changes. |
| <b>Example</b>     | Sub Main()<br><br>End Sub                                                                                                                      |

```

' Closes the active document
MsgBox "Closing " + WM.ActiveDocument.Name
WM.ActiveDocument.Close
End Sub

```

**See Also** WMDocument (object)

## WMDocument.Collide (method)

**Syntax** WMDocument.Collide [*state*]

**Description** Specifies whether the selected objects could collide.

**Comments** The WMDocument.Collide method takes the selected set of WMBody objects and controls whether the objects could collide or not. You can select or deselect objects using the Select or SelectAll method of the WMDocument. You can verify the current set of objects by accessing WMDocument.Selection property.

The method takes the following parameter:

| Parameter    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>state</i> | <p>An optional Boolean specifying whether the objects could collide (if True) or not collide (if False). The parameter is optional, and the default value is True.</p> <p>You can use the SelectAll method immediately followed by the Collide method to make all bodies non-collidable. If objects other than WMBody are included in the selection, they will be ignored.</p> <p>By default, all WMBody objects could collide with one another. The exception is made when two bodies are directly connected with joints or gears. For more detail on collision properties, please refer to Chapter 4 of the <i>Working Model User's Manual</i>.</p> |

**Example**

```

Sub Main()
' Makes all bodies in the active document not collidable.
Dim D as WMDocument
Set D = WM.ActiveDocument
D.SelectAll ' Selects all objects, including non-WMBody
D.Collide False
End Sub

```

**See Also** WMDocument (object), WMDocument.Select (method), WMDocument.Selection (property)

---

**WMDocument.CombineTapeScroll (property)**


---

**Syntax** WMDocument.CombineTapeScroll

**Description** A Boolean to show whether the tape control display is combined in line with the horizontal scroll bar or displayed in parallel with the scroll bar.

**Comments** CombineTapeScroll is True if the Tape Control is displayed in line with the horizontal scroll bar. When the property is set to False, the Tape Control is displayed in parallel to the horizontal scroll bar. Changing the combine status will activate the Save menu item.

**Example**

```
Sub Main()
 ' Combines the tape control with horizontal scroll bar.
 ' Warns the user if it is already so.

 Dim D as WMDocument
 Set D = WM.ActiveDocument

 if D.CombineTapeScroll = True then
 MsgBox "Tape Control is already combined"
 else
 D.CombineTapeScroll = True
 end if
End Sub
```

**See Also** WMDocument (object)

---

**WMDocument.Constraint (method)**


---

**Syntax** WMDocument.Constraint(*name* | *id*)

**Description** Returns the first WMConstraint object in the document that matches the given name or ID number, or the special value *Nothing* if none is found.

**Comments** The WMDocument.Constraint method takes one or the other of the following parameters:

| Parameter   | Description                                                                                                          |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>name</i> | A String containing the name of the WMConstraint object to be searched. The string match will not be case-sensitive. |
| <i>id</i>   | An Integer specifying the ID of the WMConstraint.                                                                    |

**Example** Sub Main()

```

' Outputs the name of constraint[3].

Dim D as WMDocument

Dim C as WMConstraint

Set D = WM.ActiveDocument

Set C = D.Constraint(3)

If C is not Nothing Then

 MsgBox "Constraint(3) is called "+C.Name

End If

End Sub

```

**See Also** WMDocument (object), WMConstraint (object)

## WMDocument.Constraints (property)

---

**Syntax** WMDocument.Constraints

**Description** Returns the collection of all WMConstraint objects present in the document.

**Comments** The Constraints property is a collection of all WMConstraint objects present in the document. Like any other Collection object, you can use the Item method to access a specific constraint within the collection.

The Constraints property is read-only.

The index parameter given to the Item method is sequential within the set of constraints in the document. For example, if a document Doc has 10 objects (bodies, constraints, etc.) and 3 of the 10 objects are constraints, these constraints may be referred to as Doc.Constraint(3), Doc.Constraint(6) and Doc.Constraint(7) with the Constraint method. With the Constraints property, these objects are referred to as Doc.Constraints.Item(1), Doc.Constraints.Item(2), and Doc.Constraints.Item(3), but not necessarily in that order. The Constraints property is provided as a convenient tool to access all constraints in a loop statement (see Example), and the indices given to the Item are not permanently linked to individual WMConstraint objects.

**Example** Sub Main()

```

' hide all the constraints in the document

Dim D as WMDocument

Set D = WM.ActiveDocument

For I = 1 to D.Constraints.Count

 D.Constraints.Item(I).Show = False

```

Next

End Sub

**See Also** WMDocument (object), WMConstraint (object), Collection (topic)

---

## WMDocument.ControlsLocked (property)

---

**Syntax** WMDocument.ControlsLocked

**Description** A Boolean to show whether the Lock Controls is turned on in the Working Model document.

**Comments** ControlsLocked is True if the controls and meters are locked in place; False otherwise. You can set ControlsLocked to True or False whether any Control object (controls and meters) exists in the Working Model document. If the flag is set to True, you can create Controls, but you will not be able to move them until you disable Lock Controls through the menu or by setting ControlsLocked to False.

**Example**

```
Sub Main()
 ' Toggles Lock Control on the active document.

 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ControlsLocked = True then
 D.ControlsLocked = False
 else
 D.ControlsLocked = True
 end if
End Sub
```

**See Also** WMDocument (object)

---

## WMDocument.Copy (method)

---

**Syntax** WMDocument.Copy

**Description** Copies selected set of objects in the Working Model document to the Clipboard.

**Comments** The objects to be copied need to be selected first. The Copy method has no effect if no object is selected.

The method is equivalent of the Copy menu item in the Edit menu. The selected objects will not be modified.

**Example**

```
Sub Main()
```

```
' Creates a square and duplicates it on the screen.
Dim D as WMDocument
Dim B as WMBody
Set D = WM.ActiveDocument
' Creates a square; by default, located at (0, 0)
Set B = D.NewBody("square")
B.Width.Value = 1.0
D.Select B
D.Copy
' Duplicates a square.
D.Paste
' Now we will select both squares, "cut" them,
' and paste them.
D.SelectAll
D.Cut
D.Paste
End Sub
```

**See Also** WMDocument (object), WMDocument.Select (method),  
WMDocument.SelectAll (method), WMDocument.Cut (method),  
WMDocument.Paste (method)

---

## **WMDocument.CurrentFrame (property)**

---

**Syntax** WMDocument.CurrentFrame

**Description** An Integer containing the current frame number of the simulation.

**Comments** The CurrentFrame is an Integer property which contains the current frame number of the simulation. If no simulation history is stored, or simulation is reset, CurrentFrame always contains 0.

CurrentFrame is a read-only property.

To obtain the number of frames stored in memory for the current document, use the WMDocument.HistoryFrames property.

**Example** (See WMDocument.Reset)

**See Also** WMDocument (object), WMDocument.HistoryFrames (property),  
WMDocument.Run (method), WMDocument.Reset (method),  
WMDocument.StartHere (method)

---

## WMDocument.Cut (method)

---

|                    |                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMDocument.Cut</code>                                                                                                                                                                          |
| <b>Description</b> | Copies a selected set of objects in the Working Model document to the Clipboard and deletes the objects from the document.                                                                           |
| <b>Comments</b>    | The objects to be copied need to be selected first.<br><br>The method is equivalent of the Cut menu item in the Edit menu. The selected objects will be deleted from the document.                   |
| <b>Example</b>     | See <code>WMDocument.Copy</code> .                                                                                                                                                                   |
| <b>See Also</b>    | <code>WMDocument</code> (object), <code>WMDocument.Select</code> (method), <code>WMDocument.SelectAll</code> (method), <code>WMDocument.Copy</code> (method), <code>WMDocument.Paste</code> (method) |

---

## WMDocument.DecimalDigits (property)

---

|                    |                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMDocument.DecimalDigits</code>                                                                                                              |
| <b>Description</b> | An Integer which specifies the number of decimal digits displayed in Working Model's meters and dialogs.                                           |
|                    | <b>Note:</b> This property is not related to <code>WMDocument.SignificantDigits</code> .                                                           |
| <b>Comments</b>    | The property has different significance depending on the current setting of <code>WMDocument.DecimalFormat</code> .<br><br>The default value is 3. |
| <b>Example</b>     | See <code>WMDocument.DecimalFormat</code> .                                                                                                        |
| <b>See Also</b>    | <code>WMDocument.DecimalFormat</code> (property), <code>WMDocument.UnitSystem</code> (property)                                                    |

---

## WMDocument.DecimalFormat (property)

---

| <b>Syntax</b>      | <code>WMDocument.DecimalFormat</code>                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | A String which specifies the display format for floating point numbers in Working Model's meters and dialogs.                                                                                                                                                                                                        |
| <b>Comments</b>    | <code>DecimalFormat</code> can take one of the three values as follows. Note that the display format also depends on the setting of the <code>DecimalDigits</code> property of the <code>WMDocument</code> object. Please refer to the <i>Working Model User's Manual</i> for more information on Numbers and Units. |
| <b>Value</b>       | <b>Description</b>                                                                                                                                                                                                                                                                                                   |
| Auto               | Working Model automatically adjusts the number display to show easy-to-read numbers. The number of digits after the floating point is exactly the                                                                                                                                                                    |



one specified in `DecimalDigits`. For example, 0.000123 and 333.3333 are displayed as 1.2e-4 and 333.3, respectively, provided `DecimalDigits = 1`.

**Floating** All numbers are displayed in the form of `x.YYYeZ`, where `x` is always one digit, and the number of sub-decimal digits (i.e., how many `ys` to be displayed) is given by `DecimalDigits`. `Z`, the exponent, is always an integer. For example, 0.000123 and 333.3333 are displayed as 1.2e-4 and 3.3e2, respectively, provided `DecimalDigits = 1`.

**Fixed** All numbers are displayed without exponents (no matter how large or small it is), with as many sub-decimal digits as specified in `DecimalDigits`. For example, 0.000123 and 333.3333 are displayed as 0.0 and 333.3, respectively, provided `DecimalDigits = 1`.

The default value is "Auto".

**Example**

```
Sub Main()
 ' Set the number format to fixed with 5 digits.
 WM.ActiveDocument.DecimalFormat = "Fixed"
 WM.ActiveDocument.DecimalDigits = 5
End Sub
```

**See Also** `WMDocument.DecimalDigits` (property), `WMDocument.UnitSystem` (property)

## WMDocument.Delete (method)

**Syntax** `WMDocument.Delete [object]`

**Description** Deletes a selected set of objects in the Working Model document.

**Comments** The optional parameter, *object*, can be a `WMBody`, `WMConstraint`, `WMPoint`, `WMOutput`, or `WMInput`. If the parameter is provided, the `Delete` method will delete the object.

If no parameter is specified, the `Delete` method deletes the currently selected set of objects. If no object is selected, the `Delete` method has no effect.

**Example**

```
Sub Main()
 ' Delete all objects in the document
 WM.ActiveDocument.SelectAll
 WM.ActiveDocument.Delete
End Sub
```

**See Also** `WMDocument` (object), `WMDocument.Select` (method), `WMDocument.SelectAll` (method)

---

**WMDocument.DeletePauseControl (method)**


---

| <b>Syntax</b>      | <code>WMDocument.DeletePauseControl index</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Deletes an existing pause control condition.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>Comments</b>    | The <code>DeletePauseControl</code> takes the following parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Parameter          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <i>index</i>       | <p>An Integer specifying the pause condition. The value of <i>index</i> can be between 1 and <code>PauseControlCount</code> inclusive.</p> <p>Please see the section on <code>WMDocument.NewPauseControl</code> for how pause conditions are created.</p> <p>Deleting a pause condition decrements the <code>PauseControlCount</code> property by 1 (provided there is at least 1 pause condition before the method is called). If a pause condition is deleted, other conditions whose number was higher than <i>index</i> will be shifted down and assigned a new number (decremented by 1).</p> <p>For example, suppose three pause conditions exist. If you invoke:</p> <pre>Doc.DeletePauseControl 2</pre> <p>then the condition 3 will change its number to 2, while the condition 1 remains unchanged. No condition 3 exists anymore thereafter. See the Example below.</p> |

**Example**

```
Sub Main()
 ' Creates three pause conditions and deletes the second.
 ' Assumes no pause condition exists heretofore.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 D.NewPauseControl
 D.NewPauseControl
 D.NewPauseControl
 D.PauseControl(1).Formula = "time > 1"
 D.PauseControl(2).Formula = "time > 2"
 D.PauseControl(3).Formula = "time > 3"
 D.DeletePauseControl 2
 ' At this point, the condition "time > 3" is reassigned
 ' as Condition 2.
 MsgBox "Condition 2 is now:" + D.PauseControl(2).Formula
```

```
' The message box displays "time > 3"
End Sub
```

**See Also** WMDocument (object), WMDocument.NewPauseControl (method),  
WMDocument.PauseControlCount (property),  
WMDocument.SetPauseControlType (method),  
WMDocument.GetPauseControlType (method),  
WMDocument.PauseControl (method)

### WMDocument.DistanceUnit (property)

|                    |                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.DistanceUnit                                                                                                                                                                 |
| <b>Description</b> | A String which specifies the distance unit in the Working Model document.                                                                                                               |
| <b>Comments</b>    | The DistanceUnit property can have one of the following values.                                                                                                                         |
| <b>Value</b>       | <b>Unit Description</b>                                                                                                                                                                 |
| Angstroms          | Angstroms.                                                                                                                                                                              |
| Centimeters        | Centimeters.                                                                                                                                                                            |
| Feet               | Feet.                                                                                                                                                                                   |
| Inches             | Inches.                                                                                                                                                                                 |
| Kilometers         | Kilometers.                                                                                                                                                                             |
| Light Years        | Light years.                                                                                                                                                                            |
| Meters             | Meters.                                                                                                                                                                                 |
| Micrometers        | Micrometers (microns).                                                                                                                                                                  |
| Miles              | Miles.                                                                                                                                                                                  |
| Millimeters        | Millimeters.                                                                                                                                                                            |
| Mils               | Mils (one thousandth of an inch).                                                                                                                                                       |
| Nanometers         | Nanometers.                                                                                                                                                                             |
| Parsecs            | Parsecs.                                                                                                                                                                                |
| Yards              | Yards.                                                                                                                                                                                  |
|                    | Both lower- and upper-case letters are accepted. The default value of the DistanceUnit property is "Meters".                                                                            |
|                    | The property is overwritten when the user explicitly specifies the UnitSystem property of the document, because each unit system has a set of specifications for all measurement units. |

**Example** See `WMDocument.UnitSystem`.

**See Also** `WMDocument.UnitSystem` (property), `WMDocument.DecimalFormat` (property), `WMDocument.DecimalDigits` (property)

---

## ***WMDocument.ElectricPotentialUnit* (property)**

---

**Syntax** `WMDocument.ElectricPotentialUnit`

**Description** A `String` which specifies the electric potential unit in the Working Model document.

**Comments** The `ElectricPotentialUnit` property can have one of the following values.

| Value  | Unit Description                                                                                                                                                                                                                                                                                                                                           |
|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Volts  | Volts.                                                                                                                                                                                                                                                                                                                                                     |
| (null) | None. (i.e., <code>ElectricPotentialUnit = ""</code> )<br><br>The default value of the <code>ElectricPotentialUnit</code> property is "Volts".<br><br>The property is overwritten when the user explicitly specifies the <code>UnitSystem</code> property of the document, because each unit system has a set of specifications for all measurement units. |

**Example** See `WMDocument.UnitSystem`.

**See Also** `WMDocument.UnitSystem` (property), `WMDocument.DecimalFormat` (property), `WMDocument.DecimalDigits` (property)

---

## ***WMDocument.ElectrostaticConst* (property)**

---

**Syntax** `WMDocument.ElectrostaticConst`

**Description** A `Double` which specifies the electrostatic constant in the Working Model document.

**Comments** You must turn on the electrostatics first in order for this constant to take effect in the Working Model document (see `WMDocument.ElectrostaticsOn`). The property `ElectrostaticConst` pertains to the value of the constant  $1/4 \epsilon_0$  used in computing  $F = (C_1 C_2 / r^2) / 4 \epsilon_0$ , where  $F$  is the force acting on every pair of bodies (whose charge is  $C_1$  and  $C_2$  and located a distance  $r$  apart) in the document.  
  
The default value of `ElectrostaticConst` is  $8.99 \times 10^9 \text{ Nm}^2/\text{C}^2$ . The numerical value is automatically adjusted to accommodate the current unit system in order to retain the physical magnitude of the constant.

**Example** (See `WMDocument.ElectrostaticsOn`)

**See Also** WMDocument (object), WMDocument.ElectrostaticsOn (property)

## WMDocument.ElectrostaticsOn (property)

| <b>Syntax</b>      | WMDocument.ElectrostaticsOn                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | A Boolean which specifies whether the electrostatic forces are turned on in the Working Model document.                                                                                                          |
| <b>Comments</b>    | Electrostatic forces act between each pair of bodies according to their charge. You can specify the value of <code>ElectrostaticsOn</code> as follows.                                                           |
| Value              | Unit Description                                                                                                                                                                                                 |
| True               | Electrostatic force is on.                                                                                                                                                                                       |
| False              | Electrostatic force is off.                                                                                                                                                                                      |
|                    | Use the property <code>WMDocument.ElectrostaticConst</code> to modify the value of multiplicative constant $1/4 \epsilon_0$ .                                                                                    |
| <b>Example</b>     | <pre>Sub Main()     ' Turns on Electrostatics.     Dim D as WMDocument : Set D = WM.ActiveDocument     D.ElectroStaticsOn = True     D.ElectrostaticConst = 8.990e+9     MsgBox D.Electrostaticson End Sub</pre> |

**See Also** WMDocument (object), WMDocument.ElectrostaticConst (property)

## WMDocument.EnergyUnit (property)

| <b>Syntax</b>      | WMDocument.EnergyUnit                                                     |
|--------------------|---------------------------------------------------------------------------|
| <b>Description</b> | A String which specifies the energy unit in the Working Model document.   |
| <b>Comments</b>    | The <code>EnergyUnit</code> property can have one of the following values |
| Value              | Unit Description                                                          |
| Joules             | Joules (J).                                                               |
| Kilowatt hours     | Kilowatt hours (kWh).                                                     |
| Kilocalories       | Kilocalories (Kcal).                                                      |
| B. T. U.           | British thermal units (BTU).                                              |
| Electron Volts     | Electron volts (eV).                                                      |

## 578 Working Model Basic User's Manual

---

|                 |                                                                                                                                                                                                                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MeV             | Megaelectron volts (MeV).                                                                                                                                                                                                                                                                                                                 |
| Ergs            | Ergs (Erg).                                                                                                                                                                                                                                                                                                                               |
| (null)          | None. (i.e., EnergyUnit = "")                                                                                                                                                                                                                                                                                                             |
|                 | Both lower- and upper-case letters are accepted. The default value of the DistanceUnit property is "Joules".                                                                                                                                                                                                                              |
|                 | The property is overwritten when the user explicitly specifies the UnitSystem property of the document, because each unit system has a set of specifications for all measurement units.                                                                                                                                                   |
|                 | When EnergyUnit is set to "" (null), Working Model displays the energy unit as a composite unit based on the setting in ForceUnit and DistanceUnit. For example, when DistanceUnit is "Meters", ForceUnit is "Newtons", and EnergyUnit is "" (null), then meters and Properties window show the velocity unit as "N m", or Newton-meters. |
| <b>Example</b>  | See WMDocument.UnitSystem.                                                                                                                                                                                                                                                                                                                |
| <b>See Also</b> | WMDocument.UnitSystem (property), WMDocument.DecimalFormat (property), WMDocument.DecimalDigits (property)                                                                                                                                                                                                                                |

### **WMDocument.EraseMeterValues (method)**

---

|                    |                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.EraseMeterValues                                                                                                                                                                                                                                                                                           |
| <b>Description</b> | Flushes from memory the meter data taken from multiple simulation runs.                                                                                                                                                                                                                                               |
| <b>Comments</b>    | <p>When the RetainMeterValues property of the document is True, Working Model stores the meter data from multiple simulations into its memory. Use the EraseMeterValues method to flush the memory.</p> <p>The EraseMeterValues method is only valid when the RetainMeterValues property of the document is True.</p> |
| <b>Example</b>     | <pre>Sub Main()<br/>    ' Erase meter values only when the Retain Meter Values<br/>    ' feature is active<br/><br/>    Dim D as WMDocument : Set D = WM.ActiveDocument<br/>    If D.RetainMeterValues then<br/>        D.EraseMeterValues<br/>    End If<br/>End Sub</pre>                                           |
| <b>See Also</b>    | WMDocument.RetainMeterValues (property)                                                                                                                                                                                                                                                                               |

---

## WMDocument.EraseTrack (method)

---

- Syntax** WMDocument.EraseTrack
- Description** Erases traced tracks left on the Working Model document window.
- Comments** Use the EraseTrack method to refresh the simulation tracks.

**Example**

```
Sub Main()
 ' Erase tracks only if AutoEraseTrack is off, and
 ' Tracking is turned on.

 Dim D as WMDocument
 Set D = WM.ActiveDocument

 If (D.AutoEraseTrack = False and D.Tracking <> 0) then
 D.EraseTrack
 End if
End Sub
```

**See Also** WMDocument.AutoEraseTrack (property)

---

## WMDocument.ExportDXF (method)

---

- Syntax** WMDocument.ExportDXF *filename*
- Description** Exports the document to a DXF file.
- Comments** The method takes the following parameter.

| Parameter | Description |
|-----------|-------------|
|-----------|-------------|

|                 |                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filename</i> | A String which specifies the name of the DXF file.<br><br>Please refer to the <i>Working Model User's Manual</i> for details regarding how DXF files are created from objects in Working Model. |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Example**

```
Sub Main()
 ' Opens a file and exports the model to a dxf file.

 Dim D as WMDocument
 If FileExists("4bar.wm") then
 Set D = WM.Open("4bar.wm")
 D.ExportDXF "4bar.dxf"
 else
 MsgBox "Error: File 4bar.wm does not exist."
```

```
end if
End Sub
```

**See Also** WMDocument (object), WMDocument.ImportDXF (method)

---

## WMDocument.ExportMeterData (method)

---

| <b>Syntax</b>      | WMDocument.ExportMeterData <i>filename</i> [ <i>MeterSelect</i> ]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Exports the meter data to a text file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Comments</b>    | The method takes the following parameters.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Parameter          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <i>filename</i>    | A String which specifies the name of the meter data file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <i>MeterSelect</i> | <p>An optional Boolean parameter which specifies whether to export the data from selected meters only or from all meters. When set True, Working Model will export meter data only from selected meters. Use WMDocument.Select method to select meters and objects. The default value is False.</p> <p>By default, frame 0 up to the last frame currently stored in the simulation history will be exported to a meter data file. To specify the initial and final frames for the meter data export, set ExportStartFrame and ExportStopFrame properties of the WMDocument object.</p> |

**Example**

```
Sub Main()
 ' If time history exists, and meter exists,
 ' exports meter data from all meters.
 Dim D as WMDocument : Set D = WM.ActiveDocument
 if D.HistoryFrames > 0 and D.Outputs.Count > 0 then
 D.ExportStartFrame = 0
 D.ExportStopFrame = D.HistoryFrames
 D.ExportMeterData "Datafile.dta"
 end if
End Sub
```

**See Also** WMDocument (object), WMDocument.ExportStartFrame (property), ExportStopFrame (property)

---

## WMDocument.ExportStartFrame (property)

---

**Syntax** WMDocument.ExportStartFrame



|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Specifies the initial frame number when exporting a data file.                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Comments</b>    | <p>The <code>ExportStartFrame</code> is an <code>Integer</code> property which specifies the initial frame number for exporting files. Working Model will export the data file (such as Meter Data), starting from the frame designated by <code>ExportStartFrame</code>, finishing the export (and the simulation) with the <code>ExportStopFrame</code>.</p> <p>The default value of <code>ExportStartFrame</code> is 0.</p> |
| <b>Example</b>     | (See <code>WMDocument.ExportMeterData</code> )                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>See Also</b>    | <code>WMDocument</code> (object), <code>WMDocument.ExportMeterData</code> (method), <code>ExportStopFrame</code> (property)                                                                                                                                                                                                                                                                                                    |

## ***WMDocument.ExportStopFrame (property)***

---

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMDocument.ExportStopFrame</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Description</b> | Specifies the final frame number when exporting a data file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Comments</b>    | <p>The <code>ExportStopFrame</code> is an <code>Integer</code> property which specifies the final frame number for exporting files. Working Model will export the data file (such as Meter Data), starting from the frame designated by <code>ExportStartFrame</code>, finishing the export (and the simulation) with the <code>ExportStopFrame</code>.</p> <p>The default value of <code>ExportStopFrame</code> is 100 (if no simulation history exists) or the last frame currently stored in the simulation history.</p> |
| <b>Example</b>     | (See <code>WMDocument.ExportMeterData</code> )                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>See Also</b>    | <code>WMDocument</code> (object), <code>WMDocument.ExportMeterData</code> (method), <code>ExportStartFrame</code> (property)                                                                                                                                                                                                                                                                                                                                                                                                |

## ***WMDocument.ForceFieldFX, ForceFieldFY, ForceFieldT (properties)***

---

|                    |                                                                                                                                                                         |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMDocument.ForceFieldFX</code> , <code>WMDocument.ForceFieldFY</code> , <code>WMDocument.ForceFieldT</code>                                                       |
| <b>Description</b> | <code>WMCCell</code> properties which specify the X-, Y-, and torque components of the force field in the Working Model document, respectively                          |
| <b>Comments</b>    | <p>You must activate the force field first in order for these properties to take effect in the Working Model document (see <code>WMDocument.ForceFieldType</code>).</p> |
| <b>Example</b>     | (See <code>WMDocument.ForceFieldType</code> )                                                                                                                           |
| <b>See Also</b>    | <code>WMDocument</code> (object), <code>WMDocument.ForceFieldType</code> (property)                                                                                     |

## WMDocument.ForceFieldType (property)

| <b>Syntax</b>      | WMDocument.ForceFieldType                                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | A String which specifies the type of the force field in the Working Model document.                                                                                                                                                                                                                                                                      |
| <b>Comments</b>    | The ForceFieldType property can have one of the following values.                                                                                                                                                                                                                                                                                        |
| <b>Value</b>       | <b>Unit Description</b>                                                                                                                                                                                                                                                                                                                                  |
| off                | Off. No active force field.                                                                                                                                                                                                                                                                                                                              |
| pairwise           | Force field acts between each pair of bodies in the document.                                                                                                                                                                                                                                                                                            |
| field              | Force field acts uniformly on the document.<br><br>Use WMDocument.ForceFieldFX, ForceFieldFY, and ForceFieldT properties to specify the individual components of the force field.<br><br>The force field acts on bodies in the document <i>in addition to</i> forces set by AirResistanceType, Gravity, and ElectrostaticsOn properties of the document. |
| <b>Example</b>     | <pre> Sub Main()     ' Sets up magnetic force field pointing into the     ' simulation window.      Dim D as WMDocument : Set D = WM.ActiveDocument      D.ForceFieldType = "field"      D.ForceFieldFX.Formula = "-1e4 * self.charge * self.v.y"     D.ForceFieldFY.Formula = "1e4 * self.charge * self.v.x"  End Sub </pre>                            |
| <b>See Also</b>    | WMDocument (object), WMDocument.ForceFieldFX, ForceFieldFY, ForceFieldT (properties)                                                                                                                                                                                                                                                                     |

## WMDocument.ForceUnit (property)

| <b>Syntax</b>      | WMDocument.ForceUnit                                                   |
|--------------------|------------------------------------------------------------------------|
| <b>Description</b> | A String which specifies the force unit in the Working Model document. |
| <b>Comments</b>    | The ForceUnit property can have one of the following values.           |
| <b>Value</b>       | <b>Unit Description</b>                                                |
| Dynes              | Dynes.                                                                 |
| Grams (force)      | Gram forces.                                                           |
| Kilograms (force)  | Kilogram forces.                                                       |

|                 |                                                                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Newton          | Newton.                                                                                                                                                                                 |
| Poundal         | Poundal.                                                                                                                                                                                |
| Pound           | Pound.                                                                                                                                                                                  |
|                 | Both lower- and upper-case letters are accepted. The default value of the ForceUnit property is "Newtons".                                                                              |
|                 | The property is overwritten when the user explicitly specifies the UnitSystem property of the document, because each unit system has a set of specifications for all measurement units. |
| <b>Example</b>  | See WMDocument.UnitSystem.                                                                                                                                                              |
| <b>See Also</b> | WMDocument.UnitSystem (property), WMDocument.DecimalFormat (property), WMDocument.DecimalDigits (property)                                                                              |

## WMDocument.FrequencyUnit (property)

|                    |                                                                                                                                                                                                                                          |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.FrequencyUnit                                                                                                                                                                                                                 |
| <b>Description</b> | A String which specifies the frequency unit in the Working Model document.                                                                                                                                                               |
| <b>Comments</b>    | The FrequencyUnit property can take one of the following.                                                                                                                                                                                |
| <b>Value</b>       | <b>Unit Description</b>                                                                                                                                                                                                                  |
| Hertz              | Hertz (Hz).                                                                                                                                                                                                                              |
| (null)             | None. (i.e., FrequencyUnit = "")                                                                                                                                                                                                         |
|                    | The default value of the FrequencyUnit property is "" (null).                                                                                                                                                                            |
|                    | Both lower- and upper-case letters are accepted. The property is overwritten when the user explicitly specifies the UnitSystem property of the document, because each unit system has a set of specifications for all measurement units. |
| <b>Example</b>     | See WMDocument.UnitSystem.                                                                                                                                                                                                               |
| <b>See Also</b>    | WMDocument.UnitSystem (property), WMDocument.DecimalFormat (property), WMDocument.DecimalDigits (property)                                                                                                                               |

## WMDocument.GetPauseControlType (method)

|                    |                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.GetPauseControlType (index)                                                      |
| <b>Description</b> | Returns the String specifying the action to be taken when the pause condition is satisfied. |
| <b>Comments</b>    | The SetPauseControlType method takes the following parameter.                               |

## 584 Working Model Basic User's Manual

---

| Parameter    | Description                                                                                                                         |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>index</i> | An Integer specifying the pause condition. The value of <i>index</i> can be between 1 and <code>PauseControlCount</code> inclusive. |

The return value of the `GetPauseControlType` method can be one of the following:

| Value   | Description                                                                                                                                                                                              |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "pause" | Pauses the simulation when the condition is satisfied. The user can click the Run button to continue the simulation.                                                                                     |
| "stop"  | Stops the simulation when the condition is satisfied. After the simulation is stopped with this pause condition, the user must remove or modify this pause condition to continue the simulation further. |
| "reset" | Resets the simulation to frame 0 when the condition is satisfied.                                                                                                                                        |
| "loop"  | Loops the simulation when the condition is satisfied. The simulation will be repeated indefinitely.                                                                                                      |

To specify the condition, use the `SetPauseControlType` method.

For more information on pause control, please see the section on `WMDocument.NewPauseControl` method.

### Example

```
Sub Main()
 ' Checks all the current pause conditions, and switches
 ' them all to Reset When.

 Dim D as WMDocument
 Set D = WM.ActiveDocument
 for I = 1 to D.PauseControlCount
 if D.GetPauseControlType(I) <> "reset" then
 D.SetPauseControlType I, "reset"
 end if
 next
End Sub
```

### See Also

`WMDocument` (object), `WMDocument.NewPauseControl` (method)  
`WMDocument.PauseControlCount` (property), `WMDocument.PauseControl`  
(method), `WMDocument.SetPauseControlType` (method),  
`WMDocument.DeletePauseControl` (method)

## ***WMDocument.Gravity* (property)**

---

**Syntax** `WMDocument.Gravity`

|                    |                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | A String which controls the type of gravitational forces acting in the Working Model document.                                                                                                                            |
| <b>Comments</b>    | The Gravity property can have one of the following values.                                                                                                                                                                |
| <b>Value</b>       | <b>Unit Description</b>                                                                                                                                                                                                   |
| none               | No gravity.                                                                                                                                                                                                               |
| linear             | Gravity acts uniformly on every body in the document. The force acts in the negative y-direction. See the property <code>WMDocument.LinearGravityConst</code> to modify the gravitational constant.                       |
| planetary          | Gravity acts between each pair of bodies. The magnitude of the gravity is determined by Newton's law of gravitation. See the property <code>WMDocument.PlanetaryGravityConst</code> to modify the gravitational constant. |

The default value is linear.

**Example**

```
Sub Main()
 ' Sets up the lunar gravitational field.
 Dim D as WMDocument : Set D = WM.ActiveDocument
 D.Gravity = "linear"
 D.LinearGravityConst = 1.67
End Sub
```

**See Also** `WMDocument` (object), `WMDocument.LinearGravityConst` (property), `WMDocument.PlanetaryGravityConst` (property)

## ***WMDocument.HistoryFrames(property)***

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | <code>WMDocument.HistoryFrames</code>                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>Description</b> | An Integer containing the number of simulation frames currently stored in memory.                                                                                                                                                                                                                                                                                                                                            |
| <b>Comments</b>    | <p>The <code>HistoryFrames</code> is an Integer property which contains the number of frames currently stored in memory for the given document. If no simulation history is stored, <code>HistoryFrames</code> always contains 0.</p> <p><code>HistoryFrames</code> is a read-only property.</p> <p>To obtain the current frame number displayed in the document, use the <code>WMDocument.CurrentFrame</code> property.</p> |
| <b>Example</b>     | (See <code>WMDocument.ExportMeterData</code> )                                                                                                                                                                                                                                                                                                                                                                               |
| <b>See Also</b>    | <code>WMDocument</code> (object), <code>WMDocument.CurrentFrame</code> (property), <code>WMDocument.Run</code> (method), <code>WMDocument.Reset</code> (method), <code>WMDocument.StartHere</code> (method)                                                                                                                                                                                                                  |

---

## WMDocument.ImportDXF (method)

---

| <b>Syntax</b>      | <code>WMDocument.ImportDXF filename</code>                                                                                                                                                   |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Imports a DXF file into the document.                                                                                                                                                        |
| <b>Comments</b>    | The method takes the following parameter.                                                                                                                                                    |
| Parameter          | Description                                                                                                                                                                                  |
| <i>filename</i>    | A String which specifies the name of the DXF file.<br><br>Please refer to <i>Working Model User's Manual</i> for details regarding how DXF files are translated to objects in Working Model. |

**Example**

```
Sub Main()
 ' Imports a DXF file
 Dim D as WMDocument : Set D = WM.ActiveDocument
 If FileExists("4bar.dxf") then
 D.ImportDXF "4bar.dxf"
 End if
End Sub
```

**See Also** WMDocument (object), WMDocument.ImportDXF (method)

---

## WMDocument.Input (method)

---

| <b>Syntax</b>      | <code>WMDocument.Input(name   id)</code>                                                                                         |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | Returns the first WMInput object that matches the given name or ID number, or the special value <i>Nothing</i> if none is found. |
| <b>Comments</b>    | The WMDocument.Input method takes one or the other of the following parameters:                                                  |
| Parameter          | Description                                                                                                                      |
| <i>name</i>        | A String containing the name of the WMInput object to be searched. The string match will not be case-sensitive.                  |
| <i>id</i>          | An Integer specifying the ID of the WMInput.                                                                                     |

**Example**

```
Sub Main()
 ' Outputs the name of input[2].
 Dim D as WMDocument
 Dim I as WMInput
 Set D = WM.ActiveDocument
```

```
Set I = D.Input(2)

If I is not Nothing then

 MsgBox I.Name

End If

End Sub
```

**See Also** WMDocument (object), WMInput (object)

---

## WMDocument.Inputs (property)

---

**Syntax** WMDocument.Inputs

**Description** Returns the collection of all WMInput objects present in the document.

**Comments** The Inputs property is a collection of all WMInput objects present in the document. Like any other Collection objects, you can use the Item method to access a specific input within the collection.

The Inputs property is read-only.

The index parameter given to the Item method is sequential within the set of input objects in the document. For example, if a document Doc has 10 objects (bodies, constraints, etc.) and 3 of the 10 objects are inputs, these inputs may be referred to as Doc.Input(3), Doc.Input(6) and Doc.Input(7) with the Input method. With the Inputs property, these objects are referred to as Doc.Inputs.Item(1), Doc.Inputs.Item(2), and Doc.Inputs.Item(3), but not necessarily in that order. The Inputs property is provided as a convenient tool to access all input objects in a loop statement (see Example), and the indices given to the Item are not permanently linked to individual WMInput objects.

**Example**

```
Sub Main()

 ' change all the constraints to sliders

 Dim D as WMDocument

 Set D = WM.ActiveDocument

 For I = 1 to D.Inputs.Count

 D.Inputs.Item(I).Format = "slider"

 Next

End Sub
```

**See Also** WMDocument (object), WMInput (object), Collection (topic)

---

## WMDocument.Integrator (property)

---

**Syntax** WMDocument.Integrator

**Description** Specifies the current integrator for the `WMDocument` object.

**Comments** `Integrator` is a `String` property. It can have one of the following values.

| Parameter                 | Description                                                                                                                                                                     |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>euler</code>        | The Euler method.                                                                                                                                                               |
| <code>kutta_merson</code> | The Predictor/corrector method.<br><br>All documents have <code>kutta_merson</code> as the default simulation mode (which is set in the <code>Accurate</code> simulation mode). |

**Example**

```
Sub Main()
 ' Sets the integrator of the active document to
 ' the Euler method.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 D.Integrator = "euler"
End Sub
```

**See Also** `WMDocument` (object), `WMDocument.SimulationMode` (property)

---

## ***WMDocument.IntegratorError* (property)**

---

**Syntax** `WMDocument.IntegratorError`

**Description** Specifies the Integrator Error of the Working Model document.

**Comments** `IntegratorError` is a `Double` property. The property is exactly equivalent to the Integrator Error in the Accuracy dialog of Working Model.

For a new document, the default value of `IntegratorError` is determined by Working Model automatically. The value varies according to the size and unit system of the model.

In order to specify `IntegratorError`, you must first set `AutoIntegratorError` to `False`.

**Example**

```
Sub Main()
 ' Uses Fast simulation mode for the active document, and
 ' set the Integrator Error to 0.01
 WM.ActiveDocument.SimulationMode = "Fast"
 ' At this point, AutoIntegratorError is True and
 ' IntegratorError has the value automatically determined by
```



```

' Working Model.

WM.ActiveDocument.AutoIntegratorError = False

WM.ActiveDocument.IntegratorError = 0.05

End Sub

```

**See Also** WMDocument (object), WMDocument.AutoIntegratorError (property)

## WMDocument.Join (method)

**Syntax** WMDocument.Join

**Description** Join objects that are selected and ready to be joined in the document.

**Comments** The WMDocument.Join performs the Join operation on objects that are selected and ready to be joined in the document. For example, two individual point elements, when selected, can be joined to form a pin joint. Also constraints that are currently split can be rejoined using this method.

**Example**

```

Sub Main()

' Select point[5] and point[6], and joins them together
' to form a pin joint. Assumes point[5] and point[6] exist.

Dim D as WMDocument

Set D = WM.ActiveDocument

D.SelectAll False ' make sure no others are selected

D.Select D.Point(5)

D.Select D.Point(6)

D.Join

End Sub

```

**See Also** WMDocument (object), WMOBJECT (object)

## WMDocument.LinearGravityConst (property)

**Syntax** WMDocument.LinearGravityConst

**Description** A Double which specifies the linear gravity constant in the Working Model document.

**Comments** You must first turn on linear gravity in order for this constant to take effect in the Working Model document (see WMDocument.Gravity). The property LinearGravityConst pertains to the gravitational constant  $g$  used in computing  $F = mg$ , where  $F$  is the force acting on every body (whose mass is  $m$ ) in the document.

The default value of `LinearGravityConst` is 9.81 m / sec<sup>2</sup> (386 in / sec<sup>2</sup>). The numerical value is automatically adjusted to accommodate the current unit system in order to retain the physical magnitude of the constant.

See `WMDocument.PlanetaryGravityConst` to specify the gravitational constant *G* for planetary gravity.

**Example** (See `WMDocument.Gravity`)

**See Also** `WMDocument` (object), `WMDocument.Gravity` (property), `WMDocument.PlanetaryGravityConst` (property)

---

## ***WMDocument.MassUnit* (property)**

---

**Syntax** `WMDocument.MassUnit`

**Description** A `String` which specifies the mass unit in the Working Model document.

**Comments** The `MassUnit` property can have one of the following values.

| Value             | Unit Description  |
|-------------------|-------------------|
| Kilograms         | Kilograms.        |
| Earth Pounds      | British pounds.   |
| Slugs             | Slugs.            |
| Metric Tons       | Metric tons.      |
| Grams             | Grams.            |
| Milligrams        | Milligrams.       |
| Atomic mass units | Atomic mass unit. |

Both lower- and upper-case letters are accepted. The default value of the `MassUnit` property is "Kilograms".

The property is overwritten when the user explicitly specifies the `UnitSystem` property of the document, because each unit system has a set of specifications for all measurement units.

**Example** See `WMDocument.UnitSystem`.

**See Also** `WMDocument.UnitSystem` (property), `WMDocument.DecimalFormat` (property), `WMDocument.DecimalDigits` (property)

---

## ***WMDocument.Name* (property)**

---

**Syntax** `WMDocument.Name`

**Description** Returns a `String` containing the name of the Working Model document.

**Comments** The `Name` property is read-only. The property cannot be directly modified

through Working Model Basic. You can save the document under a different name using the `SaveAs` method.

**Example**

```
Sub Main()
 MsgBox "The active document name is " + WM.ActiveDocument.Name
End Sub
```

**See Also** `WMDocument (object)`, `WMDocument.SaveAs (method)`.

## WMDocument.NewBody (method)

**Syntax** `WMDocument.NewBody (type)`

**Description** Creates a new body in the Working Model document, and returns the `WMBody` object.

**Comments** The `NewBody` method takes the following parameter:

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>type</i> | <p>A <code>String</code> which specifies the type of the body to be created. The value of type can be <code>circle</code>, <code>square</code>, <code>rectangle</code>, or <code>polygon</code>.</p> <p>The default dimension of the object is based on the current grid size. For brevity, let us denote the grid size as <i>g</i>. The object will have the following default dimensions and positions.</p> <ul style="list-style-type: none"> <li>▪ A polygon will have three vertices ((0,0), (g,0), (g,g) in global coordinates) by default. Its FOR is set to the geometric center of the polygon (triangle).</li> <li>▪ A circle will have a diameter equal to <i>g</i>. The geometric center of the circle will be placed at (0, 0).</li> <li>▪ A square and rectangle will have a width and height equal to <i>g</i>. The geometric center of the body will be placed at (0, 0).</li> </ul> <p>The current grid size can be viewed by activating the grid lines in the Working Model document (use <code>WMDocument.ShowGridLines</code> property).</p> <p>After creating the object, modify the property of the new <code>WMBody</code> object to specify its initial position and geometry.</p> |

**Example**

```
Sub Main()
 ' Creates a circle of radius 2 at (x,y)=(3,3)
 Dim B1 as WMBody
 Dim Doc1 as WMDocument
 Doc1 = WM.ActiveDocument
```

```
Set B1 = Doc1.NewBody("Circle")
B1.radius.value = 2
B1.PX.Value = 3 : B1.PY.Value = 3
End Sub
```

**See Also** WMDocument (object), WMBody (object), WMDocument.ShowGridLines (property)

---

## WMDocument.NewConstraint (method)

---

**Syntax** WMDocument.NewConstraint(*type*)

**Description** Creates a new constraint in the Working Model document, and returns the corresponding WMConstraint object.

**Comments** The NewConstraint method takes the String parameter *type*, which specifies the type of the constraint to be created. The value can be one of the following.

| Value        | Description                  |
|--------------|------------------------------|
| Pin          | Pin joint.                   |
| SquarePin    | Locked joint.                |
| Spring       | Spring.                      |
| Damper       | Damper.                      |
| SpringDamper | Spring-damper combination.   |
| Rspring      | Rotational spring.           |
| Rdamper      | Rotational damper.           |
| Hslot        | Horizontal slot joint.       |
| Vslot        | Vertical slot joint.         |
| KeyedHSlot   | Keyed horizontal slot joint. |
| KeyedVSlot   | Keyed vertical slot joint.   |
| CurvedSlot   | Curved slot joint.           |
| Rod          | Rod.                         |
| Separator    | Separator.                   |
| Rope         | Rope.                        |
| Force        | Force.                       |
| Torque       | Torque.                      |

---

|          |           |
|----------|-----------|
| Actuator | Actuator. |
| Motor    | Motor.    |
| Pulley   | Pulley.   |
| Gear     | Gear.     |

When created, constraints default to the following:

- Endpoint coordinates are set to (0, 0). Slot points (base points) will be at zero.
- A curved slot has three default control points: (0, 0), (1, 1), and (2, 2).
- Endpoints are attached to the background.
- Magnitude for force and torque is set to 0.
- The `Kind` property of the constraint is set to equal to the `String` given in the *type* parameter. The only exceptions are all linear slot joints (`Vslot`, `Hslot`, `KeyedHslot`, `KeyedVslot`); these constraints have `Kind` set to "slot".

To specify the constraint properties, directly modify the properties of the new `WMConstraint` object after it is created with the `NewConstraint` method.

**Example** (See `WMConstraint`)

**See Also** `WMDocument` (object), `WMConstraint` (object)

---

## WMDocument.NewInput (method)

---

**Syntax** `WMDocument.NewInput ( )`

**Description** Creates a new input object in the Working Model document, and returns the corresponding `WMInput` object.

**Comments** To specify which object is to be controlled by the `WMInput` object, you need to modify the property of the target object itself (see Example below).

The default `Format` of the input object is `Slider`.

**Example** `Sub Main( )`

```
'Creates a square and sets up a slider bar to control its
initial x-position

Dim D as WMBody
Dim I as WMInput

Set D = WM.ActiveDocument.NewBody("square")

D.Width.Value = 1.0
```

```
Set I = WM.ActiveDocument.NewInput()
D.PX.Formula = "input[" +str$(I.ID)+ "]"
End Sub
```

**See Also** WMDocument (object), WMInput (object)

---

## WMDocument.NewOutput (method)

---

**Syntax** WMDocument.NewOutput()

**Description** Creates a new output object in the Working Model document, and returns the corresponding WMOutput object.

**Comments** To specify what quantities are to be displayed on the WMOutput object, you need to modify its properties. See Example below. Also, please refer to the section on WMOutput for more information.

The default Format of the WMOutput object is Graph.

**Example**

```
Sub Main()
 'Sets up a square and creates a meter to measure its y-
 'position.
 Dim B as WMBody
 Dim O as WMOutput
 Set B = WM.ActiveDocument.NewBody("square")
 B.Width.Value = 1.0
 Set O = WM.ActiveDocument.NewOutput()
 FLD$ = "body[" +str$(B.ID)+"].p.y"
 O.Column(1).Cell.Formula = FLD
 O.Column(1).Label = "Y-Position"
End Sub
```

**See Also** WMDocument (object), WMOutput (object)

---

## WMDocument.NewPauseControl (method)

---

**Syntax** WMDocument.NewPauseControl

**Description** Creates a new pause control condition.

**Comments** The NewPauseControl allows for an entry of a new pause control condition.

Each WMDocument object maintains a list of pause control conditions. The PauseControlCount property keeps track of how many pause control

conditions are currently implemented. The `PauseControlCount` is initially set to zero, as no pause condition is specified in a blank document.

When the `NewPauseControl` method is invoked, `PauseControlCount` is incremented by 1 and a new condition is added to the end of the list; hence the new `PauseControlCount` is equal to the position of the new condition. You must use the `PauseControl` method to specify the formula expression for the pause condition, and use the `SetPauseControlType` method to specify the action to be taken (e.g. `Pause`, `Stop`, or `Loop`) when the condition is satisfied.

Only three pause control conditions will appear in the `Pause Control` dialog, but repeated calls to `NewPauseControl` will continue to add to the end of the list.

To delete a pause control condition, use the `DeletePauseControl` method.

**Example**

```
Sub Main()
 ' Creates two pause control conditions; assume no pause
 ' conditions currently exists. Conditions are:
 ' 1: pause when frame number exceeds 100
 ' 2: reset when time exceeds 2.5.

 Dim D as WMDocument
 Set D = WM.ActiveDocument
 ' Creates two conditions in a row
 D.NewPauseControl
 D.NewPauseControl
 D.PauseControl(1).Formula = "frame() > 100"
 D.PauseControl(2).Formula = "time > 2.5"
 D.SetPauseControlType 1, "pause"
 D.SetPauseControlType 2, "reset"

End Sub
```

**See Also**

`WMDocument` (object), `WMDocument.PauseControl` (method),  
`WMDocument.PauseControlCount` (property),  
`WMDocument.SetPauseControlType` (method),  
`WMDocument.GetPauseControlType` (method),  
`WMDocument.DeletePauseControl` (method)

---

## WMDocument.NewPoint (method)

---

**Syntax** `WMDocument.NewPoint (type)`

**Description** Creates a new point element in the Working Model document, and returns the corresponding `WMPoint` object.

**Comments** The `NewPoint` method takes the following parameter:

| Parameter   | Description                                                                                                                                                                                                                                                   |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>type</i> | A <code>String</code> which specifies what type of point is to be created. The value can be either <code>Point</code> (to create a point element), <code>SquarePoint</code> (to create a square point element), or <code>Anchor</code> (to create an anchor). |

**Example**

```
Sub Main()
 ' Creates a point at (x,y)=(3,3)
 Dim P as WMPoint
 Set P = WM.ActiveDocument.NewPoint("SquarePoint")
 P.PX.Value = 3.0
 P.PY.Value = 3.0
End Sub
```

**See Also** `WMDocument` (object), `WMPoint` (object)

---

## WMDocument.Object (method)

---

**Syntax** `WMDocument.Object (name | id)`

**Description** Returns the first `WMOBJECT` object that matches the given name or ID number.

**Comments** Since the `Object` method returns a `WMOBJECT`, you must store the return value in another `WMOBJECT` variable, even if the return value is certain to be a `WMBODY` object, for example. The method `Object` is most useful when you try to apply a type-independent change to all objects in the entire document (see Example).

The `WMDocument.Object` method takes one or the other of the following parameters:

| Parameter   | Description                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name</i> | A <code>String</code> containing the name of the <code>WMOBJECT</code> object to be searched. The string match will be case-sensitive.                                 |
| <i>id</i>   | An <code>Integer</code> specifying the ID of the <code>WMOBJECT</code> .<br>The return value will be <code>Nothing</code> if no <code>WMOBJECT</code> that matches the |



criterion is found.

**Example**

```
Sub Main()
 ' Finds an object named "Body". If it is not a WMBody object,
 pops up
 ' warning for bad naming. (You can name any object in any way;
 this is
 ' a sample code to clean up naming scheme simply for the sake
 of usability)

 Dim Doc as WMDocument : Set Doc = WM.ActiveDocument

 Dim I as Integer, Obj as WMOBJECT

 Set Obj = Doc.OBJECT("body")

 If Obj is not Nothing then
 If Obj.Kind <> "body" then
 MsgBox "Object "+str$(Obj.ID)+" has a name 'body' but it
 is not!"
 End If
 End If

End Sub
```

**See Also** WMDocument (object), WMOBJECT (object)

## WMDocument.Objects (property)

**Syntax** WMDocument.Objects

**Description** Returns the collection of all WMOBJECT objects (i.e., all Working Model objects) present in the document.

**Comments** The Objects property is a collection of all WMOBJECT objects present in the document. Like any other Collection objects, you can use the Item method to access a specific object within the collection.

The Objects property is read-only.

Please note the return value of the Item method will be another WMOBJECT. For example, even if you are certain that the returned object is a WMBody, you must use WMOBJECT object to store the return value.

```
' Assume Doc is a valid WMDocument object.
' Assume body[3] exists.

Dim B as WMBody

Dim Obj as WMOBJECT

B = Doc.OBJECTS.Item(3) ' incorrect usage
```

```
obj = Doc.Objects.Item(3) ' correct usage
```

Since obj shown above is a WMOBJECT object, you can only access the properties available in WMOBJECT objects (such as Name, ShowName, etc.).

In order to access body[3] as well as its methods and properties, use the WMDocument.Body method instead.

**Example**

```
Sub Main()
 ' show the name of all the objects in the document
 Dim Doc as WMDocument : Set Doc = WM.ActiveDocument
 For I = 1 to Doc.Objects.Count
 Doc.Objects.Item(I).ShowName = True
 Next
End Sub
```

**See Also** WMDocument (object), WMOBJECT (object), Collection (topic)

---

## WMDocument.Output (method)

---

**Syntax** WMDocument.Output(name\$|id)

**Description** Returns the first WMOBJECT object that matches the given name or ID number, or the special value *Nothing* if none is found.

**Comments** The WMDocument.Output method takes one or the other of the following parameters:

| Parameter | Description                                                                                                      |
|-----------|------------------------------------------------------------------------------------------------------------------|
| name\$    | A String containing the name of the WMOBJECT object to be searched. The string match will not be case-sensitive. |
| id        | An Integer specifying the ID of the WMOBJECT.                                                                    |

**Example**

```
Sub Main()
 ' Outputs the name of output[2]. Assumes that there exists
 ' output[2].
 Dim D as WMDocument
 Dim O as WMOBJECT
 Set D = WM.ActiveDocument
 Set O = D.Output(2)
 If O is not Nothing then
 MsgBox O.Name
```

```
End If
End Sub
```

**See Also** WMDocument (object), WMOutput (object)

## WMDocument.Outputs (property)

**Syntax** WMDocument.Outputs

**Description** Returns the collection of all WMOutput objects present in the document.

**Comments** The Outputs property is a collection of all WMOutput objects present in the document. Like any other Collection objects, you can use the Item method to access a specific output within the collection.

The Outputs property is read-only.

The index parameter given to the Item method is sequential within the set of constraints in the document. For example, if a document Doc has 10 objects (bodies, constraints, etc.) and 3 of the 10 objects are outputs, these constraints may be referred to as Doc.Output(3), Doc.Output(6) and Doc.Output(7) with the Output method. With the Outputs property, these objects are referred to as Doc.Outputs.Item(1), Doc.Outputs.Item(2), and Doc.Outputs.Item(3), but not necessarily in that order. The Outputs property is provided as a convenient tool to access all outputs in a loop statement (see Example), and the indices given to the Item are not permanently linked to individual WMOutput objects.

**Example**

```
Sub Main()
 ' change all the outputs to digital meters
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 For I = 1 to D.Outputs.Count
 D.Outputs.Item(I).Format = "digital"
 Next
End Sub
```

**See Also** WMDocument (object), WMOutput (object), Collection (topic)

## WMDocument.OverlapError (property)

**Syntax** WMDocument.OverlapError

**Description** Specifies the Overlap Error of the Working Model document.

**Comments** OverlapError is a Double property. The property is equivalent to the Overlap Error in the Accuracy dialog of Working Model.

For a new document, the default value of `OverlapError` is determined by Working Model automatically. The value varies according to the size and unit system of the model.

In order to specify `OverlapError`, you must first set `AutoOverlapError` to `False`.

**Example**

```
Sub Main()
 ' Uses Fast simulation mode for the active document, and
 ' set the Overlap Error to 0.01
 WM.ActiveDocument.SimulationMode = "Fast"
 ' At this point, AutoOverlapError is True and
 ' OverlapError has the value automatically determined by
 ' Working Model.
 WM.ActiveDocument.AutoOverlapError = False
 WM.ActiveDocument.OverlapError = 0.1
End Sub
```

**See Also** `WMDocument (object)`, `WMDocument.AutoOverlapError (property)`

---

## ***WMDocument.Paste (method)***

---

**Syntax** `WMDocument.Paste`

**Description** Pastes a selected set of objects from the Clipboard to the Working Model document.

**Comments** The pasted location is automatically determined by Working Model.  
The method is equivalent of the Paste menu item in the Edit menu.

**Example** (See `WMDocument.Copy`).

**See Also** `WMDocument (object)`, `WMDocument.Select (method)`,  
`WMDocument.SelectAll (method)`, `WMDocument.Cut (method)`,  
`WMDocument.Copy (method)`

---

## ***WMDocument.PauseControl (method)***

---

**Syntax** `WMDocument.PauseControl (index)`

**Description** Returns the `WMCell` object representing the specified pause condition.

**Comments** The `PauseControl` method takes the following parameter.

| Parameter | Description |
|-----------|-------------|
|-----------|-------------|

---

|              |                                                                             |
|--------------|-----------------------------------------------------------------------------|
| <i>index</i> | An Integer specifying the pause condition. The value of <i>index</i> can be |
|--------------|-----------------------------------------------------------------------------|

between 1 and PauseControlCount inclusive.

Each WMDocument object maintains a list of pause control conditions. You can use the PauseControl method to specify the formula expressions for each pause condition. Use SetPauseControlType method to specify the action to be taken (e.g. Pause, Reset, Loop) when the condition is satisfied.

For more information on pause control, please see the section on WMDocument.NewPauseControl method.

**Example**

```
Sub Main()
 ' Creates a new pause control condition. If 3 or more
 conditions exist,
 ' does not allow further additions.

 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.PauseControlCount < 3 then
 D.NewPauseControl

 D.PauseControl(D.PauseControlCount).Formula = "time > 2.5"
 D.SetPauseControlType D.PauseControlCount, "reset"
 else
 MsgBox "Document already has 3 pause conditions!"
 end if
End Sub
```

**See Also** WMDocument (object), WMDocument.NewPauseControl (method)  
WMDocument.PauseControlCount (property),  
WMDocument.SetPauseControlType (method),  
WMDocument.GetPauseControlType (method),  
WMDocument.DeletePauseControl (method)

---

## **WMDocument.PauseControlCount (property)**

---

|                    |                                                                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.PauseControlCount                                                                                                                                                                                                                                                     |
| <b>Description</b> | Contains the current number (Integer) of pause control conditions. Read-only.                                                                                                                                                                                                    |
| <b>Comments</b>    | Each WMDocument maintains a list of pause control conditions. The PauseControlCount property keeps track of how many pause control conditions are currently implemented. The PauseControlCount is initially set to zero, as no pause condition is specified in a blank document. |

When the `NewPauseControl` method is invoked, `PauseControlCount` is incremented by 1; the new `PauseControlCount` serves as the index for the new condition. You must use the `PauseControl` method to specify the formula expression for the pause condition, and use the `SetPauseControlType` method to specify the action to be taken (e.g. Pause, Reset, Stop, or Loop) when the condition is satisfied.

To delete a pause control condition, use the `DeletePauseControl` method. The `DeletePauseControl` method decrements the `PauseControlCount` by 1.

**Example** (See `WMDocument.PauseControl`)

**See Also** `WMDocument` (object), `WMDocument.NewPauseControl` (method),  
`WMDocument.PauseControl` (method),  
`WMDocument.SetPauseControlType` (method),  
`WMDocument.GetPauseControlType` (method),  
`WMDocument.DeletePauseControl` (method)

---

## ***WMDocument.PlanetaryGravityConst* (property)**

---

**Syntax** `WMDocument.PlanetaryGravityConst`

**Description** A `Double` which specifies the planetary gravity constant in the Working Model document.

**Comments** You must turn on the planetary gravity first in order for this constant to take effect in the Working Model document (see `WMDocument.Gravity`). The property `PlanetaryGravityConst` pertains to the gravitational constant  $G$  used in computing  $F = Gm_1m_2/r^2$ , where  $F$  is the force acting on every pair of bodies (whose masses are  $m_1$  and  $m_2$ , respectively) in the document.

See `WMDocument.LinearGravityConst` to specify the gravitational constant  $g$  for linear gravity.

**Example**

```
Sub Main()
 ' Sets up the planetary gravitational field.
 Dim D as WMDocument : Set D = WM.ActiveDocument
 D.Gravity = "planetary"
 MsgBox "G = "+str$(D.PlanetaryGravityConst)
End Sub
```

**See Also** `WMDocument` (object), `WMDocument.Gravity` (property),  
`WMDocument.LinearGravityConst` (property)

## WMDocument.PlayerMode (property)

---

**Syntax** WMDocument.PlayerMode

**Description** A Boolean which specifies whether the document is in Player mode.

**Comments** When the PlayerMode property is True, the document is in Player mode. In Player mode, the toolbar, scroll bars, and most of the menu items are disabled. The document *cannot* be modified interactively (without choosing the Edit Mode first); the PlayerMode property must be set to False before any interactive modification is to be made.

Using WM Basic, you can modify documents in Player mode.

The default value of the PlayerMode is False.

**Example**

```
Sub Main()
 ' Switch a document to player mode
 WM.ActiveDocument.PlayerMode = True
End Sub
```

**See Also** WMDocument (object)

## WMDocument.Point (method)

---

**Syntax** WMDocument.Point(*name* | *id*)

**Description** Returns the first WMPoint object that matches the given name or ID number, or the special value *Nothing* if none is found.

**Comments** The WMDocument.Point method takes one or the other of the following parameters:

| Parameter   | Description                                                                                                     |
|-------------|-----------------------------------------------------------------------------------------------------------------|
| <i>name</i> | A String containing the name of the WMPoint object to be searched. The string match will not be case-sensitive. |
| <i>id</i>   | An Integer specifying the ID of the WMPoint.                                                                    |

**Example**

```
Sub Main()
 ' Outputs the name of point[5]. Assumes that point[5] exists.
 Dim D as WMDocument
 Dim P as WMPoint
 Set D = WM.ActiveDocument
 Set P = D.Point(5)
```

```
 If P is not Nothing then
 MsgBox P.Name
 End If
End Sub
```

**See Also** WMDocument (object), WMPPoint (object)

---

## WMDocument.Points (property)

---

**Syntax** WMDocument.Points

**Description** Returns the collection of all WMPPoint objects present in the document.

**Comments** The Points property is a collection of all WMPPoint objects present in the document. Like any other Collection objects, you can use the Item method to access a specific point within the collection.

The Points property is read-only.

The index parameter given to the Item method is sequential within the set of points in the document. For example, if a document Doc has 10 objects (bodies, constraints, etc.) and 3 of the 10 objects are points, these points may be referred to as Doc.Point(3), Doc.Point(6) and Doc.Point(7) with the Point method. With the Points property, these objects are referred to as Doc.Points.Item(1), Doc.Points.Item(2), and Doc.Points.Item(3), but not necessarily in that order. The Points property is provided as a convenient tool to access all points in a loop statement (see Example), and the indices given to the Item are not permanently linked to individual WMPPoint objects.

**Example**

```
Sub Main()
 ' change the name of all points to "hinge"

 Dim D as WMDocument
 Dim B as WMBody
 Set D = WM.ActiveDocument
 For I = 1 to D.Points.Count
 D.Points.Item(I).Name = "hinge"
 Next
End Sub
```

**See Also** WMDocument (object), WMPPoint (object), Collection (topic)

---

## WMDocument.PowerUnit (property)

---

**Syntax** WMDocument.PowerUnit



|                    |                                                                                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | A String which specifies the power unit in the Working Model document.                                                                                                                  |
| <b>Comments</b>    | The PowerUnit property can take one of the following.                                                                                                                                   |
| <b>Value</b>       | <b>Unit Description</b>                                                                                                                                                                 |
| Watts              | Watts (W).                                                                                                                                                                              |
| Horsepower         | Horsepower (HP).                                                                                                                                                                        |
| (null)             | None. (i.e., PowerUnit = "").                                                                                                                                                           |
|                    | Both lower- and upper-case letters are accepted. The default value of the PowerUnit property is "Watts".                                                                                |
|                    | The property is overwritten when the user explicitly specifies the UnitSystem property of the document, because each unit system has a set of specifications for all measurement units. |
| <b>Example</b>     | See WMDocument.UnitSystem.                                                                                                                                                              |
| <b>See Also</b>    | WMDocument.UnitSystem (property), WMDocument.DecimalFormat (property), WMDocument.DecimalDigits (property)                                                                              |

## WMDocument.RetainMeterValues (property)

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.RetainMeterValues                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Description</b> | A Boolean to specify whether the meter data from multiple simulations are to be kept in memory.                                                                                                                                                                                                                                                                                                                                                            |
| <b>Comments</b>    | <p>When RetainMeterValues is True, Working Model keeps the meter data from multiple simulations in memory. When the property is False, the meter data is flushed every time new simulation is run. Please refer to the <i>Working Model User's Manual</i> for more information on the Retain Meter Values feature.</p> <p>The default value of RetainMeterValues is False.</p> <p>Use the EraseMeterValues method of the document to flush the memory.</p> |
| <b>Example</b>     | (See WMDocument.EraseMeterValues)                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>See Also</b>    | WMDocument.EraseMeterValues (method)                                                                                                                                                                                                                                                                                                                                                                                                                       |

## WMDocument.Reset (method)

|                    |                                                                                |
|--------------------|--------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.Reset                                                               |
| <b>Description</b> | Resets a Working Model simulation.                                             |
| <b>Comments</b>    | WMDocument.Reset has no effect if the simulation has not run or is at frame 0. |

**Example**

```
Sub Main()
 ' Erase History. Make sure that the current frame is 0.
 Dim D as WMDocument : Set D = WM.ActiveDocument
 If D.CurrentFrame <> 0 then
 if MsgBox("Resetting first; ok?", vbOKCancel) = vbOK then
 D.Reset
 D.StartHere
 end if
 else
 D.Reset
 D.Starthere
 end if
End Sub
```

**See Also** WMDocument (object), WMDocument.Reset (method)

---

## ***WMDocument.RotationalVelocityUnit* (property)**

---

**Syntax** WMDocument.RotationalVelocityUnit

**Description** A String which specifies the angular velocity unit in the Working Model document.

**Comments** The RotationalVelocityUnit property can have one of the following values.

| Value    | Unit Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Revs/Min | Revolutions per minute.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| (null)   | None. (i.e., RotationalVelocityUnit = "")<br><br>The default value of the RotationalVelocityUnit property is an empty string ("", meaning null). When RotationalVelocityUnit is set to null, Working Model displays the velocity unit as a composite unit based on the setting in RotationUnit and TimeUnit. For example, when RotationUnit is "Radians", TimeUnit is "Seconds", and RotationalVelocityUnit is null, then meters and Properties window show the velocity unit as "rad/s", or radians per second.<br><br>Both lower- and upper-case letters are accepted. The property is overwritten when the user explicitly specifies the UnitSystem property of the document, because each unit system has a set of specifications for all measurement units. |

**Example** See `WMDocument.UnitSystem`.

**See Also** `WMDocument.UnitSystem (property)`, `WMDocument.DecimalFormat (property)`, `WMDocument.DecimalDigits (property)`

## WMDocument.RotationUnit (property)

**Syntax** `WMDocument.RotationUnit`

**Description** A String which specifies the rotation unit in the Working Model document.

**Comments** The `RotationUnit` property can take one of the following values.

| Value       | Unit Description |
|-------------|------------------|
| Degrees     | Degrees.         |
| Radians     | Radians.         |
| Seconds     | Seconds.         |
| Minutes     | Minutes.         |
| Revolutions | Revolutions.     |

Both lower- and upper-case letters are accepted. The default value of the `RotationUnit` property is "Degrees".

The property is overwritten when the user explicitly specifies the `UnitSystem` property of the document, because each unit system has a set of specifications for all measurement units.

**Example** See `WMDocument.UnitSystem`.

**See Also** `WMDocument.UnitSystem (property)`, `WMDocument.DecimalFormat (property)`, `WMDocument.DecimalDigits (property)`

## WMDocument.Run (method)

**Syntax** `WMDocument.Run [frames]`

**Description** Runs a Working Model simulation.

**Parameter Comments**

|               |                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>frames</i> | <p>An Integer parameter specifying the number of frames to be run. The simulation will stop after computing the frame (<i>frames</i>)-1.</p> <p>If the parameter <i>frames</i> is omitted, the simulation will run indefinitely. If Pause Control conditions are set, the simulation will stop as soon as computing the frame (<i>frames</i>)-1 or at least one of the pause conditions (if specified) is satisfied.</p> |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

You can use the `CurrentFrame` property to access the current frame number when the simulation is paused.

---

**Note:** The execution control is given to Working Model until the `Run` method has finished. To avoid running a never-ending script, we recommend providing the parameter *frames* and/or setting Pause Control conditions (see `WMDocument.NewPauseControl`).

---

**Example**

```
Sub Main()
 ' Runs the simulation for 25 frames and saves the result with
 history.

 Dim D as WMDocument
 Set D = WM.ActiveDocument

 D.Run 25 ' Working Model starts the simulation.
 ' The remainder of the code is not executed until frame 24
 ' is calculated.

 D.Reset

 D.SaveAs "modell.wm", True ' saves the history, too.

End Sub
```

**See Also**

`WMDocument` (object), `WMDocument.Reset` (method),  
`WMDocument.CurrentFrame` (property), `WMDocument.NewPauseControl`  
(method)

---

## ***WMDocument.Save* (method)**

---

**Syntax** `WMDocument.Save`

**Description** Saves the Working Model document.

**Comments** If the document has not been previously saved (i.e., the document is still “untitled”), Working Model will display a dialog prompting the filename as well as folder/directory location. To avoid the confirmation dialog, use the `SaveAs` method.

**Example**

```
Sub Main()
 ' Saves the active document if it has a meaningful name
 ' other than "Untitled".

 If InStr(WM.ActiveDocument.Name, "Untitled") then
 MsgBox "Use SaveAs first to name the document."
 Else
 MsgBox "Saving " + WM.ActiveDocument.Name + "."
 End If
End Sub
```

```

 WM.ActiveDocument.Save
 End If
End Sub

```

**See Also** WMDocument (object), WMDocument.SaveAs (method)

## WMDocument.SaveAs (method)

| <b>Syntax</b>         | WMDocument.SaveAs <i>filename</i> [ , <i>isHistorySaved</i> ]                                                                                 |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | Saves the Working Model document under the specified filename.                                                                                |
| <b>Comments</b>       | The SaveAs method takes the following parameters.                                                                                             |
| Parameter             | Description                                                                                                                                   |
| <i>filename</i>       | A String to specify the filename of the document is to be saved.                                                                              |
| <i>isHistorySaved</i> | (optional) A Boolean to specify whether the simulation history is to be saved. The history will be saved if True. The default value is False. |

**Example**

```

Sub Main()
 'Save the current file with time history
 Dim WM1 as WMDocument : Set WM1 = WM.ActiveDocument
 If Basic.OS = ebMacintosh then
 WM1.SaveAs "My Important Model", True
 ElseIf Basic.OS = ebWin16 then
 WM1.SaveAs "model1.wm", True
 End If
End Sub

```

**See Also** WMDocument (object), WMDocument.Save (method)

## WMDocument.ScaleFactor (property)

|                    |                                                                                                                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.ScaleFactor                                                                                                                                                                                                                                                                                                        |
| <b>Description</b> | A Double to specify the scale factor of the Working Model document shown on the screen.                                                                                                                                                                                                                                       |
| <b>Comments</b>    | <p>ScaleFactor indicates how large or small the Working Model document is displayed on the screen.</p> <p>The ScaleFactor property is inversely proportional to the ViewWidth property of the document. Since the document pixel size is not affected by modifying the ScaleFactor property, ViewWidth will automatically</p> |

double when `ScaleFactor` is halved, for example.

Please refer to the *Working Model User's Manual* for more information.

**Example**

```
Sub Main()
 ' Sets the view size so that it displays 10 meters wide
 ' Also, focuses the origin at the center, and reports
 ' the current scale factor on the window.

 Dim D as WMDocument : Set D = WM.ActiveDocument
 D.UnitSystem = "si degrees"
 D.ViewWidth = 10.0
 D.ScrollTo 0, 0

 MsgBox "Current Scale Factor: "+str$(D.ScaleFactor)

End Sub
```

**See Also** `WMDocument.ViewWidth` (property)

---

## ***WMDocument.ScrollTo* (method)**

---

**Syntax** `WMDocument.ScrollTo x, y`

**Description** Scrolls the document window to focus in a particular area of the document.

**Comments** You can use the `ScrollTo` method to shift the focus of the document display to the desired position. The method takes the following parameters.

| Parameter         | Description                                                                                                                                                                                                                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>x, y</code> | ( <code>x, y</code> ) global coordinates in the document.<br><br>The document will be scrolled so that ( <code>x, y</code> ) becomes the center of the document.<br><br>You can also modify <code>ScaleFactor</code> and <code>ViewSize</code> properties to change the outlook of the document. |

**Example** (See `WMDocument.ScaleFactor`)

**See Also** `WMDocument.ScaleFactor` (property), `WMDocument.ViewWidth` (property)

---

## ***WMDocument.Select* (method)**

---

**Syntax** `WMDocument.Select object [, state]`

**Description** Selects or de-selects the specified *object* in the document.

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Comments</b>  | The WMDocument.Select method takes the following parameters:                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Parameter</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>object</i>    | An object of type WMObject (which includes WMBody, WMConstraint, WMInput, WMOutput, and WMPoint).                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>state</i>     | <p>An Boolean specifying whether the object is to be selected (if True) or deselected (if False). The parameter is optional, and the default value is True.</p> <p>This method does the equivalent of a “shift-select” in the UI. That is, the specified object is added/removed from the collection of selected objects. Other selected objects remain selected. If you want <i>only</i> the specified object to be selected, you must first use WMDocument.SelectAll to ensure that all other objects are deselected.</p> |

**Example**

```

Sub Main()
 ' Select point[5]. Assumes that there exists point[5] in
 ' the active document.

 Dim D as WMDocument
 Dim P as WMPoint

 Set D = WM.ActiveDocument
 Set P = D.Point(5)

 D.Select P, True '"True" is optional.

End Sub

```

**See Also** WMDocument (object), WMObject (object)

## WMDocument.SelectAll (method)

|                    |                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax</b>      | WMDocument.SelectAll [ <i>state</i> ]                                                                                                                                |
| <b>Description</b> | Selects or de-selects all objects in the document.                                                                                                                   |
| <b>Comments</b>    | The WMDocument.SelectAll method takes the following parameter:                                                                                                       |
| <b>Parameter</b>   | <b>Description</b>                                                                                                                                                   |
| <i>state</i>       | An optional Boolean specifying whether all objects are to be selected (if True) or de-selected (if False). The parameter is optional, and the default value is True. |

**Example**

```

Sub Main()
 ' De-selects all objects in the document.

 Dim D as WMDocument

```

```
Set D = WM.ActiveDocument
D.SelectAll False
End Sub
```

**See Also** WMDocument (object)

## WMDocument.Selection (property)

---

**Syntax** WMDocument.Selection

**Description** An object that represents a collection of WMObject objects selected in the document.

**Comments** WMDocument.Selection itself is a read-only property. The property is automatically updated when objects are selected or de-selected. For example, if you select two objects, Selection represents those two objects. Count will return 2, and Item(1) and Item(2) will return the objects selected.

The Selection property further has the following properties and methods.

| Property                | Description                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Count                   | A read-only Integer that contains the number of objects currently selected. The number will increase / decrease as you select more / less objects (see WMDocument.Select and WMDocument.SelectAll).                                                                                                                                                                                       |
| Method                  | Description                                                                                                                                                                                                                                                                                                                                                                               |
| Item ( <i>n</i> )       | Returns the <i>n</i> th object in the selection. The returned object may be a WMBody, WMConstraint, WMPoint, WMOutput, or WMInput. This property can only be assigned to a variable of type WMObject. This method is only of use in looping over all selected objects, regardless of their type and checking, or modifying, some common property at the WMObject level (like name or id). |
| Body ( <i>n</i> )       | Returns the <i>n</i> th WMBody object in the selection. If there are fewer than <i>n</i> WMBody objects selected, the return value will be nothing.                                                                                                                                                                                                                                       |
| Constraint ( <i>n</i> ) | Returns the <i>n</i> th WMConstraint object in the selection. If there are fewer than <i>n</i> WMConstraint objects selected, the return value will be nothing.                                                                                                                                                                                                                           |
| Point ( <i>id</i> )     | Returns the <i>n</i> th WMPoint object in the selection. If there are fewer than <i>n</i> WMPoint objects selected, the return value will be nothing.                                                                                                                                                                                                                                     |
| Input ( <i>n</i> )      | Returns the <i>n</i> th WMInput object in the selection. If there are fewer than <i>n</i> WMInput objects selected, the return value will be nothing.                                                                                                                                                                                                                                     |



**Output (*n*)** Returns the *n*th WMOutput object in the selection. If there are fewer than *n* WMOutput objects selected, the return value will be nothing.

**Example**

```
Sub Main()
 ' Picks the first object in the document and determines
 ' its type. If the object is not body, constraint, or point,
 yields
 ' a message. (In that case, the object could be an input or
 output.)

 Dim B as WMBody
 Dim C as WMConstraint
 Dim P as WMPoint
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 D.SelectAll

 If D.Selection.Item(1) is Nothing Then
 ' no object exists in the document
 Exit Sub
 End If

 If D.Selection.Item(1).Kind = "body" Then
 Set B = D.Selection.Body(1)
 ElseIf D.Selection.Item(1).Kind = "constraint" Then
 Set C = D.Selection.Constraint(1)
 ElseIf D.Selection.Item(1).Kind = "point" Then
 Set P = D.Selection.Point(1)
 Else
 MsgBox "Object 1 is neither body, constraint, nor point!"
 End If
End Sub
```

**See Also** WMObject (object), WMDocument.Select (method), WMDocument.SelectAll (method)

---

## **WMDocument.SetPauseControlType (method)**

---

**Syntax** WMDocument.SetPauseControlType *index*, *condition*

**Description** Specifies the action to be taken when the indexed pause condition is

satisfied.

**Comments** The `SetPauseControlType` method takes the following parameter.

| Parameter        | Description                                                                                                                         |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>index</i>     | An Integer specifying the pause condition. The value of <i>index</i> can be between 1 and <code>PauseControlCount</code> inclusive. |
| <i>condition</i> | A String which specifies the action. See below.                                                                                     |

The parameter *condition* takes one of the following values:

| Value   | Description                                                                                                                                                                                                                                                               |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| "pause" | Pauses the simulation when the condition is satisfied. The user can click the Run button to continue the simulation.                                                                                                                                                      |
| "stop"  | Stops the simulation when the condition is satisfied. The user cannot click the Run button to continue the simulation. After the simulation is stopped with this pause condition, the user must remove or modify this pause condition to continue the simulation further. |
| "reset" | Resets the simulation to frame 0 when the condition is satisfied.                                                                                                                                                                                                         |
| "loop"  | Loops the simulation when the condition is satisfied. The simulation will be repeated indefinitely.                                                                                                                                                                       |

To retrieve the condition, use the `GetPauseControlType` method.

For more information on pause control, please see the section on `WMDocument.NewPauseControl` method.

**Example** (See `WMDocument.PauseControl`)

**See Also** `WMDocument` (object), `WMDocument.NewPauseControl` (method), `WMDocument.PauseControlCount` (property), `WMDocument.PauseControl` (method), `WMDocument.GetPauseControlType` (method), `WMDocument.DeletePauseControl` (method)

---

## ***WMDocument.ShowCoordinates* (property)**

---

**Syntax** `WMDocument.ShowCoordinates`

**Description** A Boolean to show whether the Coordinates bar is turned on in the Working Model document.

**Comments** `ShowCoordinates` is `True` if the coordinates bar is displayed; `False` otherwise. Changing the Coordinates bar display status will enable the Save menu item.

**Example**

```
Sub Main()
 ' Show Coordinates bar. Notifies the user if it is already on.
```

```
Dim D as WMDocument
Set D = WM.ActiveDocument
if D.ShowCoordinates = True then
 MsgBox "Coordinates bar is already on"
else
 D.ShowCoordinates = True
end if
End Sub
```

**See Also** WMDocument (object)

---

## WMDocument.ShowGridLines (property)

---

**Syntax** WMDocument.ShowGridLines

**Description** A Boolean to show whether the grid lines are displayed in the Working Model document.

**Comments** ShowGridLines is True if the grid lines are displayed; False otherwise. Changing the grid line display status will enable the Save menu item.

**Example**

```
Sub Main()
 ' Show grid lines. Warns the user if they are already on.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ShowGridLines = True then
 MsgBox "Grid lines are already on"
 else
 D.ShowGridLines = True
 end if
End Sub
```

**See Also** WMDocument (object)

---

## WMDocument.ShowHelpRibbon (property)

---

**Syntax** WMDocument.ShowHelpRibbon

**Description** A Boolean to show whether the Help Ribbon is displayed in the Working Model document.

**Comments** ShowHelpRibbon is True if the Help Ribbon is displayed; False otherwise. Changing the Help Ribbon display status will enable the Save

menu item.

**Example**

```
Sub Main()
 ' Hide Help Ribbon. Warns the user if it is already off.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ShowHelpRibbon = False then
 MsgBox "Help Ribbon is already off"
 else
 D.ShowHelpRibbon = False
 end if
End Sub
```

**See Also** WMDocument (object)

---

**WMDocument.ShowRulers (property)**

---

**Syntax**

WMDocument.ShowRulers

**Description**

A Boolean to show whether the rulers are displayed in the Working Model document.

**Comments**

ShowRulers is True if the rulers are displayed; False otherwise. Changing the ruler display status will activate the Save menu item.

**Example**

```
Sub Main()
 ' Show Rulers. Warns the user if it is already on.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ShowRulers = True then
 MsgBox "Rulers are already on"
 else
 D.ShowRulers = True
 end if
End Sub
```

**See Also** WMDocument (object)

---

**WMDocument.ShowScrollBars (property)**

---

**Syntax**

WMDocument.ShowScrollBars

**Description** A Boolean to show whether the scroll bars are displayed in the Working Model document.

**Comments** ShowScrollBars is True if the rulers are displayed; False otherwise. Changing the scroll bars display status will enable the Save menu item.

**Example**

```
Sub Main()
 ' Hide scroll bars. Warns the user if they are already off.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ShowScrollBars = False then
 MsgBox "Scroll bars are already turned off"
 else
 D.ShowScrollBars = False
 end if
End Sub
```

**See Also** WMDocument (object)

---

## **WMDocument.ShowTapeControl (property)**

---

**Syntax** WMDocument.ShowTapeControl

**Description** A Boolean to show whether the tape control is displayed in the Working Model document.

**Comments** ShowTapeControl is True if the Tape Control is displayed; False otherwise. Changing the Tape Control display status will enable the Save menu item.

**Example**

```
Sub Main()
 ' Hide Tape Control. Warns the user if it is already off.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ShowTapeControl = False then
 MsgBox "Tape Control is already off"
 else
 D.ShowTapeControl = False
 end if
End Sub
```

**See Also** WMDocument (object)

---

**WMDocument.ShowToolPalette (property)**

---

**Syntax** WMDocument.ShowToolPalette

**Description** A Boolean to show whether the Toolbar is displayed in the Working Model document.

**Comments** ShowToolPalette is True if the Toolbar is displayed; False otherwise. Changing the Toolbar display status will enable the Save menu item.

**Example**

```
Sub Main()
 ' Hide Tool Palette. Warns the user if it is already off.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ShowToolPalette = False then
 MsgBox "Tool palette is already off"
 else
 D.ShowToolPalette = False
 end if
End Sub
```

**See Also** WMDocument (object)

---

**WMDocument.ShowXYAxes (property)**

---

**Syntax** WMDocument.ShowXYAxes

**Description** A Boolean to show whether the global x- and y-axes are displayed in the Working Model document.

**Comments** ShowXYAxes is True if the axes are displayed; False otherwise. Changing the axes display status will enable the Save menu item.

**Example**

```
Sub Main()
 ' Show XY Axes. Warns the user if it is already on.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 if D.ShowXYAxes = True then
 MsgBox "Axes are already on"
 else
 D.ShowXYAxes = True
 end if
```

End Sub

**See Also** WMDocument (object)

## WMDocument.SignificantDigits (property)

**Syntax** WMDocument.SignificantDigits

**Description** Specifies the Significant Digits of the Working Model document.

**Comments** SignificantDigits is an Integer property. The property is equivalent to the Significant Digits in the Accuracy dialog of Working Model. The value of SignificantDigits can range from 1 to 16. Attempts to assign an out-of-range value are silently ignored while the value of SignificantDigits remains unchanged.

For a new document, the default value of SignificantDigits is determined by Working Model automatically. The value varies according to the size and unit system of the model.

In order to specify SignificantDigits, you must first set AutoSignificantDigits to False.

**Example**

```
Sub Main()
 ' Uses Fast simulation mode for the active document, and
 ' set the Significant Digits to 10
 WM.ActiveDocument.SimulationMode = "Fast"
 ' At this point, AutoSignificantDigits is True and
 ' SignificantDigits has the value automatically determined by
 ' Working Model.
 WM.ActiveDocument.AutoSignificantDigits = False
 WM.ActiveDocument.SignificantDigits = 10
End Sub
```

**See Also** WMDocument (object), WMDocument.AutoSignificantDigits (property)

## WMDocument.SimulationMode (property)

**Syntax** WMDocument.SimulationMode

**Description** Specifies the current simulation mode for the WMDocument object.

**Comments** SimulationMode is a String property. Setting the property is exactly equivalent to choosing the simulation mode in the Accuracy dialog of Working Model. All other simulation parameters (such as time step and active warnings) are automatically determined when the simulation mode is specified to be either Fast or Accuracy. Please refer to the *Working*

*Model User's Manual* for more information.

The property can be one of the following. In particular, the Fast and Accurate modes have predefined parameters for the simulation accuracy.

| Parameter | Description                                                           |
|-----------|-----------------------------------------------------------------------|
| fast      | The Fast mode. Employs the Euler method as the integrator.            |
| accurate  | The Accurate mode. Employs the Kutta-Merson method as the integrator. |
| custom    | No default values.                                                    |

All documents have `Accurate` as the default simulation mode.

**Example**

```
Sub Main()
 ' Sets the simulation mode of the active document to
 ' the Fast mode.
 Dim D as WMDocument
 Set D = WM.ActiveDocument
 D.SimulationMode = "Fast"
End Sub
```

**See Also** `WMDocument` (object), `WMDocument.Integrator` (property)

---

## ***WMDocument.SkipFrames* (property)**

---

**Syntax** `WMDocument.SkipFrames`

**Description** An Integer to specify the frame skip rate used in animation.

**Comments** `SkipFrames` specifies the rate of frame refresh while animating the simulation result.

The default value of `SkipFrames` is 1, indicating that every frame is displayed. When set to 2, for example, the animation will skip every other frame.

**Example**

```
Sub Main()
 ' Change the skip frame rate of the active document to 4
 WM.ActiveDocument.SkipFrames = 4
End Sub
```

**See Also** `WMDocument` (object)

---

## ***WMDocument.Split* (method)**

---

**Syntax** `WMDocument.Split`



- Description** Split objects that are selected and joined in the document.
- Comments** The `WMDocument.Split` performs the Split operation on objects that are selected and ready to be split in the document. For example, a pin joint can be split into two point elements. If a body is selected and it is currently attached to something with a joint, then that joint will be split also.
- Example**
- ```
Sub Main()
    ' Pins a rectangle the background, splits the pin joint
    ' (just to show the effect of split), and runs the simulation.
    Dim D as WMDocument : Set D = WM.ActiveDocument
    Dim Rect as WMBody
    Dim Joint as WMConstraint
    Set Rect = D.NewBody("rectangle")
    Rect.Width.Value = 1.0 : Rect.Height.Value = 2.0
    Set Joint = D.NewConstraint("pin")
    Set Joint.Point(2).Body = Rect
    D.SelectAll False ' Deselect all objects first
    D.Select Rect      ' Select the rectangle
    D.Split
    D.Run 25           ' the rectangle will fall
    D.Reset
End Sub
```
- See Also** `WMDocument (object)`, `WMOBJECT (object)`

WMDocument.StartHere (method)

- Syntax** `WMDocument.StartHere`
- Description** Erases the simulation history stored in memory, and sets the current frame to be the initial frame for the next simulation.
- Comments** The `StartHere` method has no effect on the current frame. The method simply erases all the simulation history (if any), and sets the current frame (contained in the `CurrentFrame` property of the document) as the initial frame (frame 0).
- As a result of the `StartHere` method, both `HistoryFrames` and `CurrentFrame` properties will be set to 0.
- Example** (See `WMDocument.Reset`)

See Also `WMDocument` (object), `WMDocument.Run` (method), `WMDocument.Reset` (method), `WMDocument.CurrentFrame` (property), `WMDocument.HistoryFrames` (property)

***WMDocument.TimeUnit* (property)**

Syntax	<code>WMDocument.TimeUnit</code>
Description	A <code>String</code> which specifies the time unit in the Working Model document.
Comments	The <code>TimeUnit</code> property can have one of the following values.
Value	Unit Description
<code>Seconds</code>	Seconds (with <code>s</code> as unit display for meters).
<code>Seconds (sec)</code>	Seconds (with <code>sec</code> as unit display for meters).
<code>Minutes</code>	Minutes.
<code>Hours</code>	Hours (with <code>h</code> as unit display for meters).
<code>Hours (hr)</code>	Hours (with <code>hr</code> as unit display for meters).
<code>Milliseconds</code>	Milliseconds.
<code>Years</code>	Years.
<code>Microseconds</code>	Microseconds.
<code>Days</code>	Days.
	Both lower- and upper-case letters are accepted. The default value of the <code>TimeUnit</code> property is "Seconds".
	The property is overwritten when the user explicitly specifies the <code>UnitSystem</code> property of the document, because each unit system has a set of specifications for all measurement units.
Example	See <code>WMDocument.UnitSystem</code> .
See Also	<code>WMDocument.UnitSystem</code> (property), <code>WMDocument.DecimalFormat</code> (property), <code>WMDocument.DecimalDigits</code> (property)

***WMDocument.Tracking* (property)**

Syntax	<code>WMDocument.Tracking</code>
Description	An <code>Integer</code> to specify the tracking rate used in animation.
Comments	The <code>Tracking</code> property specifies the rate at which a traced track is displayed in the Working Model document. The default value of <code>Tracking</code> is 0, indicating that no traced track is displayed. When set to 1, for example, the animation will leave traced

track every frame.

Example

```
Sub Main()
    ' track every 6 frames in the current document
    Dim Brick as WMBody
    Dim Doc as WMDocument : Set Doc = WM.ActiveDocument
    Doc.Gravity = "linear"
    Set Brick = Doc.NewBody("square")
    Brick.Mass.Value = 1.0
    WM.ActiveDocument.Tracking = 6
    WM.ActiveDocument.Run 100 ' run for 100 frames and see
End Sub
```

See Also WMDocument.SkipFrames (property)

WMDocument.UnitSystem (property)

Syntax WMDocument.UnitSystem

Description A String which specifies the unit system currently employed in the Working Model document.

Comments The UnitSystem property can be one of the following.

Value	Description
astronomical	Astronomical (e.g., light-years for disatance, years for time).
atomic	Atomic (e.g., nanometers for distance, atomic mass unit for mass).
cgs	The CGS unit system (e.g., centimeters for distance, grams for mass, seconds for time).
si degree	The SI unit system (e.g., meters for distance, kilograms for mass, seconds for time) with degrees as the angular measurement.
si radian	The SI unit system (e.g., meters for distance, kilograms for mass, seconds for time) with radians as the angular measurement.
english (pounds)	English system (e.g., inches for distance, seconds for time) with pounds as the mass measurement (with acceleration on Earth built-in; therefore 0.45 kilogram of mass is translated to 1 pound).
english (slugs)	English system (e.g., inches for distance, seconds for time) with slugs as the mass measurement.
undefined	Any unit system that does not fall into the above descriptions.

The default value is "si degrees".

Each unit system has a set of specifications for units used in various measurements. Therefore, assigning the `UnitSystem` property overrides the setting given in unit properties of the `WMDocument` object, such as `DistanceUnit`, `MassUnit`, and `TimeUnit`.

For example, if you assign the `UnitSystem` property to "english (pounds)", then `DistanceUnit`, `MassUnit`, and `TimeUnit` of the document will be changed to "inches", "pounds", and "seconds", regardless of the previous setting of these units.

If you change any of the individual measurement units, the `UnitSystem` property will be automatically changed to "undefined", reflecting the fact that the unit system may no longer conform to the one previously specified.

Example

```
Sub Main()  
    ' Starts out with English (pounds) unit system,  
    ' and modifies various unit systems.  
  
    Dim D as WMDocument  
    Set D = WM.ActiveDocument  
  
    D.UnitSystem = "english earth pounds"  
  
    ' At this point, various units are set to default; for example,  
    ' length unit is set to inches, and mass unit is set to pounds.  
    ' For details on each units, see individual sections.  
  
    D.DistanceUnit = "feet"  
    D.TimeUnit = "minutes"  
    D.MassUnit = "milligrams"  
    D.ChargeUnit = "Statcoulombs"  
    D.RotationUnit = "Revolutions"  
    D.ElectricPotentialUnit = "" ' "" means null string.  
    D.ForceUnit = "Poundals"  
    D.EnergyUnit = "Kilocalories"  
    D.PowerUnit = "Horsepower"  
    D.FrequencyUnit = "Hertz"  
    D.VelocityUnit = "Miles per Hour"  
    D.RotationalVelocityUnit = "Revs/Min"  
  
End Sub
```

See Also WMDocument.DecimalDigits (property), WMDocument.DecimalFormat (property)

WMDocument.Update (method)

Syntax WMDocument.Update

Description Updates the Working Model document screen to reflect the modifications made on any WMCell object.

Comments For performance reasons, when a WMCell object property is modified (e.g., position of a circle, dimension of a rectangle), the document screen will not be updated until the script is terminated. If you wish to see the result of a modification immediately, use the Update method to synchronize the Working Model document window with the current WMCell object settings.

Note: Any modification on WMCell is immediately recorded in the Working Model database. Computational results of a simulation is unaffected whether or not the Update method is used. The Update method is only necessary to “refresh” the document screen (i.e., for visual rendering).

You may find the method useful in debugging (to see the effect of a code step-by-step) and for presentation purposes (to demonstrate animated results for audience).

Example

```
Sub Main()
    ' Moves the circle from 0.0 to 2.0 incrementally by 0.2. The
    motion of
    ' the circle is rendered on the screen as it is repositioned
    each time.
    ' Without the Update, you will only see a circle positioned at
    (2, 0) at
    ' the end (i.e., no animation).
    Dim Doc as WMDocument, Disk as WMBody
    Set Doc = WM.ActiveDocument
    Set Disk = Doc.NewBody("circle")
    For I = 0.0 to 2.0 Step 0.2
        Disk.PX.Value = I
        Doc.Update ' comment out this line and see how the
        results differ
    Next I
End Sub
```

See Also `WMDocument` (object), `WMDocument.Reset` (method),
`WMDocument.CurrentFrame` (property)

***WMDocument.VariableIntegrationStep* (property)**

Syntax `WMDocument.VariableIntegrationStep`

Description Specifies whether the integration step is to be variable or fixed for the `WMDocument` object.

Comments `VariableIntegrationStep` is a Boolean property. Setting the property is exactly equivalent to checking the variable time step button in the Accuracy dialog of Working Model.

When the property is `True`, Working Model will automatically use the variable time step.

The value of `VariableIntegrationStep` is set automatically to `True` or `False`, when you specify the `SimulationMode` to `Accurate` or `Fast`, respectively.

Since the default `SimulationMode` is `Accurate`, the default value of `VariableIntegrationStep` is `True`.

Example

```
Sub Main()  
    ' Sets the simulation mode to Accurate mode, then sets the  
    ' integration step to fixed mode.  
    Dim D as WMDocument  
    Set D = WM.ActiveDocument  
    D.SimulationMode = "Accurate"  
    ' at this point VariableIntegrationStep is True, because  
    ' Accurate mode imposes the default setting.  
    D.VariableIntegrationStep = False  
End Sub
```

See Also `WMDocument` (object), `WMDocument.SimulationMode` (property),
`WMDocument.IntegrationStep` (property)

***WMDocument.VelocityUnit* (property)**

Syntax `WMDocument.VelocityUnit`

Description A String which specifies the velocity unit in the Working Model document.

Comments The `VelocityUnit` property can take one of three values as follows.

Value	Unit Description
Speed of Light	Speed of light (c).
Miles per Hour	Miles per hour (mph).
(null)	<p>None. (i.e., <code>VelocityUnit = ""</code>)</p> <p>The default value of the <code>VelocityUnit</code> property is null (can be specified as a null string, or <code>""</code>).</p> <p>Both lower and upper case letters are accepted. The property is overwritten when the user explicitly specifies the <code>UnitSystem</code> property of the document, because each unit system has a set of specification for all measurement units.</p> <p>When <code>VelocityUnit</code> is set to null, Working Model displays the velocity unit as a composite unit based on the setting in <code>DistanceUnit</code> and <code>TimeUnit</code>. For example, when <code>DistanceUnit</code> is "m" <code>TimeUnit</code> is "s", and <code>VelocityUnit</code> is "" (null), then meters and Properties window show the velocity unit as "m/s", or meters per second.</p>
Example	See <code>WMDocument.UnitSystem</code> .
See Also	<code>WMDocument.UnitSystem</code> (property), <code>WMDocument.DecimalFormat</code> (property), <code>WMDocument.DecimalDigits</code> (property)

WMDocument.ViewWidth (property)

Syntax	<code>WMDocument.ViewWidth</code>
Description	A <code>Double</code> to specify the length represented by the horizontal ruler of the Working Model document shown on the screen.
Comments	<p><code>ViewWidth</code> indicates the length represented by the width of the Working Model document. The number given in <code>ViewWidth</code> is interpreted in the current unit system.</p> <p>The <code>ViewWidth</code> property is inversely proportional to the <code>ScaleFactor</code> property of the document. Since the document pixel size is not affected by modifying the <code>ViewWidth</code> property, <code>ScaleFactor</code> will automatically double when <code>ViewWidth</code> is halved, for example.</p>
Example	(See <code>WMDocument.ScaleFactor</code>)
See Also	<code>WMDocument.ScaleFactor</code> (property)

WMDocument.WarnInaccurate (property)

Syntax	<code>WMDocument.WarnInaccurate</code>
Description	Specifies whether the warning dialog for Inaccurate Integration is enabled.

Comments WarnInaccurate is a Boolean property. Setting the property to True is exactly equivalent to checking the Inaccurate Integration Warning check box in the Accuracy dialog of Working Model.

When the property is True, Working Model will bring up a warning dialog box when it detects the presence of a large velocity or acceleration in the simulation, which may cause instability in the system. Please refer to the *Working Model User's Manual* for more information on this warning.

The value of WarnInaccurate is set automatically to False or True, when you specify the SimulationMode to Accurate or Fast, respectively.

Since the default SimulationMode is Accurate, the default value of WarnInaccurate is False.

Example

```
Sub Main()  
    ' Sets the simulation mode to Accurate mode, except  
    ' enable the large-velocity-or-acceleration warning.  
  
    Dim D as WMDocument  
    Set D = WM.ActiveDocument  
  
    D.SimulationMode = "Accurate"  
  
    ' at this point WarnInaccurate is False, because  
    ' Accurate mode imposes the default setting.  
  
    D.WarnInaccurate = True  
  
End Sub
```

See Also WMDocument (object), WMDocument.SimulationMode (property)

WMDocument.WarnInconsistent (property)

Syntax WMDocument.WarnInconsistent

Description Specifies whether the warning dialog for inconsistent constraints is enabled.

Comments WarnInconsistent is a Boolean property. Setting the property to True is exactly equivalent to checking the Inconsistent Constraints Warning check box in the Accuracy dialog of Working Model.

When the property is True, Working Model will bring up a warning dialog box when it detects an inconsistent constraint during the simulation. Inconsistent constraints may introduce excessive force in the simulation and render the system unstable. Please refer to the *Working Model User's Manual* for more information on this warning.

The value of WarnInconsistent is set automatically to True or False, when you specify the SimulationMode to Accurate or Fast, respectively.

Since the default SimulationMode is Accurate, the default value of WarnInconsistent is True.

Example

```
Sub Main()
    ' Sets the simulation mode to Fast mode, except
    ' enable the Inconsistent Constraints Warning.

    Dim D as WMDocument
    Set D = WM.ActiveDocument
    D.SimulationMode = "Fast"

    ' at this point WarnInconsistent is False, because
    ' Fast mode imposes the default setting.

    D.WarnInconsistent = True
End Sub
```

See Also WMDocument (object), WMDocument.SimulationMode (property), WMDocument.WarnRedundant (property)

WMDocument.WarnOverlap (property)

Syntax WMDocument.WarnOverlap

Description Specifies whether the warning dialog for Initial body Overlap is enabled.

Comments WarnOverlap is a Boolean property. Setting the property to True is exactly equivalent to checking the Initial Body Overlap Warning check box in the Accuracy dialog of Working Model.

When the property is True, Working Model will bring up a warning dialog box when it detects two or more bodies are overlapping more than a specified tolerance (set in the PositionalError property) at the first frame of the simulation. Large overlap between objects may introduce excessive force in the simulation and render the system unstable. Please refer to the *Working Model User's Manual* for more information on this warning.

The value of WarnOverlap is set automatically to True, when you specify the SimulationMode to Accurate or Fast.

Since the default SimulationMode is Accurate, the default value of WarnOverlap is True.

Example

```
Sub Main()
```

```
' In order to prevent unstable simulations,
' checks to make sure the overlap warning is on.

Dim D as WMDocument
Set D = WM.ActiveDocument
If D.WarnOverlap = False then
    MsgBox "Overlap warning should be on! Turning it on..."
    D.WarnOverlap = True
End If

End Sub
```

See Also WMDocument (object), WMDocument.SimulationMode (property),
WMDocument.PostionalError (property)

WMDocument.WarnRedundant (property)

Syntax WMDocument.WarnRedundant

Description Specifies whether the warning dialog for redundant constraints is enabled.

Comments WarnRedundant is a Boolean property. Setting the property to True is exactly equivalent to checking the Redundant Constraints Warning check box in the Accuracy dialog of Working Model.

When the property is True, Working Model will bring up a warning dialog box when it detects a redundant constraints during the simulation. Redundant constraints may introduce excessive force in the simulation and render the system unstable. Please refer to the *Working Model User's Manual* for more information on this warning.

The value of SimulationMode has no bearings to the value of the WarnRedundant property.

The default value of WarnRedundant is False.

Example

```
Sub Main()

    ' Sets the simulation mode to Fast mode, and
    ' enable the Redundant Constraints warning.

    Dim D as WMDocument
    Set D = WM.ActiveDocument

    D.SimulationMode = "Fast"

    D. WarnRedundant = True

End Sub
```

See Also WMDocument (object), WMDocument.SimulationMode (property),

`WMDocument.WarnInconsistent` (property)

WMInput (object)

Syntax `WMInput`

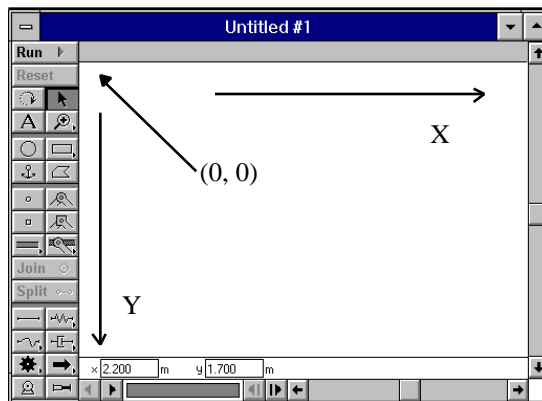
Description An object which provides an interface to input controls used in Working Model simulations.

Comments To create a new `WMInput` object in a Working Model document, use the `NewInput` method of the `WMDocument` object.

To specify which object is to be controlled by the `WMInput` object, you need to modify the properties of the target object itself (see Example below).

A `WMInput` object has the following properties.

Property	Description
<code>Kind</code>	String indicating the type of the object (<code>WMInput</code>). The property is read-only.
<code>Value</code>	Double containing the current value of the input.
<code>Format</code>	String which indicates the input format. Can be "Slider", "Table", "Button" or "TextBox". The format types correspond to the input type specified in the Properties window.
<code>Min</code>	Double to specify the minimum bound of the input. Any <code>Value</code> specified below <code>Min</code> will be truncated to <code>Min</code> .
<code>Max</code>	Double to specify the maximum bound of the input. Any <code>Value</code> specified in excess of <code>Max</code> will be truncated to <code>Max</code> .
<code>X, Y</code>	Integer to specify the (x, y) coordinates that describe the position of the <code>WMInput</code> object in the document. The values are measured in <i>pixel coordinates</i> , where the top-left corner of the document window is (0, 0). The x-coordinate increases toward the right edge of the screen, whereas the y-coordinate increases downward to the bottom of the screen.



Height, Width Integer to specify the height and width of the WMInput object in the Working Model document. As in **x** and **y** properties, **Height** and **Width** are given in pixel units.

The following properties are applicable only when **Format** is "Slider".

Property	Description
Snaps	Integer to determine the number of snaps between given Min and Max values. Working Model divides the range between Min and Max into equal-sized intervals, so that the slider only snaps at Min , Max , and (Snaps -1) steps in between. Snaps only affects the mouse-dragging of the slider. Therefore, if a WM Basic script specifies Value to be a quantity that does not fall into any of the snapping values, Working Model will accept the number as given (as long as the value is within the range specified by Min and Max).
ShowText	Boolean to indicate whether a text box will be displayed alongside the input slider. When ShowText is True , the text box will be displayed.

The following property is applicable only when **Format** is "Button".

Property	Description
Momentary	Boolean. When Momentary is set to True , the WMInput object (a button, since Format is "Button") becomes active only for the duration in which the mouse button is held down. When False , the button acts as a toggle switch. The following properties and method are applicable only when Format is "Table".

Property/method	Description
DataColumn	Integer to specify which column in the file is to be read as a data column. Default value is 1.
TimeColumn	Integer to specify which column in the file is to be read as a time

column. Default value is 0, indicating that each row of data corresponds to a single Animation Step.

ReadTable

A method to read the data table. Syntax is:

```
[WMInput].ReadTable filename
```

where the parameter *filename* is a String specifying the name of the data file.

Example

```
Sub Main()
    'Creates a square and sets the slider bar to control
    'its initial x-position.
    Dim D as WMBody
    Dim I as WMInput
    Dim id as Integer
    Set D = WM.ActiveDocument.NewBody("square")
    Set I = WM.ActiveDocument.NewInput()
    id = I.ID
    D.PX.Formula = "input[" + str$(id) + "]"
End Sub
```

See Also WMOutput (object), WMDocument.NewInput (method)

WMOBJECT (object)

Syntax WMOBJECT

Description WMOBJECT provides an interface to attributes common to on-screen objects in a Working Model document.

Comments WMOBJECT is the *parent* object of WMBody, WMConstraint, WMPoint, WMInput, and WMOOutput, in that these classes *inherit* properties/methods of WMOBJECT. An object of any of these five types will also have properties/methods of the parent.

A WMOBJECT object has the properties and methods shown below. Accordingly, objects of type WMBody, WMConstraint, WMPoint, WMInput, and WMOOutput have all these properties, except X, Y that are only available to WMInput and WMOOutput objects.

The properties X and Y control the screen location of the object and have no effect on bodies, points and constraints because the screen location is determined by scrolling (see WMDocument.ScrollTo).

The properties Width and Height are also available to WMBody objects. However, for a WMBody object, the properties are of type WMCell, and they

634 Working Model Basic User's Manual

	are evaluated in the current distance unit instead of pixels.
Property	Description
ID	<p>A read-only Integer. Every <code>WMObject</code> object is assigned a unique identification number regardless of its kind. As a general rule, the number is assigned in the order the object is created.</p> <p>The ID number is identical to the one used in Working Model's formula language (such as <code>body[5]</code>, or <code>constraint[3]</code>).</p>
Kind	<p>A read-only String. Every <code>WMObject</code> object “knows” its kind. For example, a force object (which is of type <code>WMConstraint</code>) has the <code>Kind</code> string set to “force” as soon as the object is created. This information can be used to identify the object type and define conditional statements.</p> <p>Every character in the <code>Kind</code> string is in lower-case, and the comparison needs to be case-sensitive. For instance, given a force object (<code>Kind = “force”</code>), the comparison:</p> <pre>Force.Kind = “force”</pre> <p>returns <code>True</code>, whereas:</p> <pre>Force.Kind = “Force”</pre> <p>returns <code>False</code>.</p>
Name	<p>String. Every <code>WMObject</code> object has a default name, depending on its <code>Kind</code>. For example, a body has a default name “rectangle”, “polygon”, “square” or “circle”, depending on its geometry. The <code>Name</code> property is for information only, and the user can freely modify the property.</p> <p>The <code>Name</code> string always appears on utility windows (Properties, Geometry, and Appearance). You can also display the <code>Name</code> string on the Working Model simulation by setting the <code>ShowName</code> property to <code>True</code> (see <code>ShowName</code> below).</p>
Show	<p>Boolean to indicate whether the object is visible on the Working Model simulation window. You can hide objects by setting the <code>Show</code> to <code>False</code>. The default value for <code>Show</code> is <code>True</code>.</p>
ShowName	<p>Boolean to indicate whether the <code>Name</code> property of the object is visible on the Working Model simulation window. You can display the name (custom-definable; see <code>Name</code> above) of each object when you run the simulation. The <code>Name</code> is displayed near the frame of reference of the object. The default value of <code>ShowName</code> is <code>False</code>.</p>
X, Y	<p>Integer pixel coordinates to indicate the position of the object on the screen. These properties are needed for objects of type <code>WMInput</code> and <code>WMOuput</code>, since unlike other objects in Working Model, these objects are not associated with any other coordinate system besides the pixel map on the screen.</p>

Height, Width Integer pixel size of the WMInput or WMOuput objects. (Height and Width are WMCell objects in WMBody objects.)

Method	Description
Body()	This method casts (converts) the WMOBJECT to WMBody (see Example).
Constraint()	This method casts the WMOBJECT to WMConstraint (see Example).
Input()	This method casts the WMOBJECT to WMInput (see Example).
Output()	This method casts the WMOBJECT to WMOuput (see Example).
Point()	This method casts the WMOBJECT to WMPoint (see Example).

In object-oriented programming terms, WMOBJECT may be considered as a *base class* from which WMBody and others (*derived-classes*) descend.

Example

```
Sub Main()
    'Illustrates the use of type-casts.
    ' Determines the type of the first object in the document and
    ' assigns a variable to it.
    Dim Doc as WMDocument
    Dim obj as WMOBJECT
    Dim aBody as WMBody, aPoint as WMPoint, aConstraint as
WMConstraint
    Dim anInput as WMinput, anOutput as WMOuput
    Set Doc = WM.ActiveDocument
    Doc.SelectAll
    Set obj = Doc.Selection.Item(1)
    ' Item returns a WMOBJECT
    If obj is Nothing Then
        Exit Sub
    End If
    If obj.Kind = "body" Then
        Set aBody = obj.Body() : MsgBox "Body found"
    ElseIf obj.Kind = "constraint" Then
        Set aConstraint = obj.Constraint() : MsgBox "Constraint
found"
    ElseIf obj.Kind = "point" Then
        Set aPoint = obj.Point() : MsgBox "Point found"
```

```
ElseIf obj.Kind = "output" Then
    Set anOutput = obj.Output() : MsgBox "Output found"
ElseIf obj.Kind = "input" Then
    Set anInput = obj.Input() : MsgBox "Input found"
End If
End Sub
```

See Also WMBody (object), WMConstraint (object), WMPoint (object), WMInput (object), WMOOutput (object), WMDocument.Selection (property)

WMOOutput (object)

Syntax WMOOutput

Description An object which provides an interface to output meters used in Working Model simulations.

Comments A WMOOutput object represents meters used in Working Model. Every meter in Working Model can measure up to four quantities, and so can a WMOOutput object.

Each measured quantity is specified using WMOOutputColumn objects. A WMOOutput object has methods and properties specifying how the values are displayed on the meter. Yet a WMOOutput object is not much more than a place holder for a WMOOutputColumn object. You can access individual columns of a WMOOutput object using the Column method (see below).

A WMOOutput object has the following properties and methods..

Method/PropertyDescription

Column(<i>num</i>)	A method which returns the specified WMOOutputColumn object representing the fields of the output meter. The parameter <i>num</i> of type Integer specifies the column ID. For <i>num</i> = 0, Column returns the value of the field x (x-axis). For <i>num</i> = <i>n</i> , Column returns the value of the field <i>yn</i> . Please see WMOOutputColumn section for more information.
ConnectPoints	A Boolean property to indicate whether the plotted points are to be connected.
Format	A String property to specify the display format of the Output. It can be "digital", "bar", or "graph".
Kind	A String property indicating the type of the object ; returns "output". The property is read-only.
ShowAxes	A Boolean property to indicate whether x- and y-axes are displayed when

the Output is in the graph format.

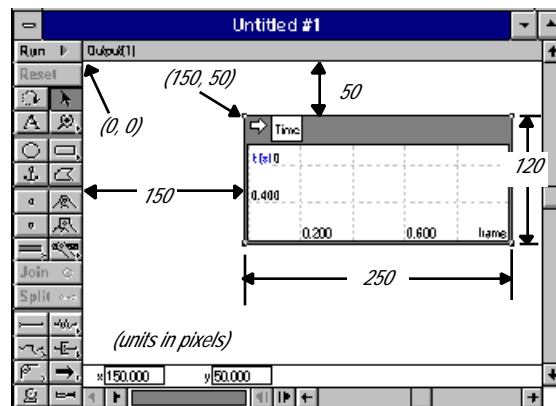
ShowGrid	Boolean to indicate whether the grid display is turned on or off.
ShowFrame	Boolean to indicate whether the Output has an outlining frame.
ShowLabels	Boolean to indicate whether to show the labels on the Output.
ShowUnits	Boolean to indicate whether the units are shown on the Output.
X, Y	Integer to specify the (x, y) coordinates that describe the position of the WMOutput object in the document. The values are measured in <i>pixel coordinates</i> , where the top-left corner of the document window is (0, 0). The x-coordinate increases toward the right edge of the screen, whereas the y-coordinate increases downward to the bottom of the screen.
H, W	Integer to specify the height and width of the WMOutput object in the Working Model document. As in x and y properties, H and W are given in pixel sizes.

For example, suppose you want to locate the WMOutput object M at (150, 50) and change its dimensions to 250 (width) by 120 (height) pixels. The code segment would be:

```
M.X = 150: M.Y = 50
M.W = 250: M.H = 120
```

(Recall a colon (:) is used to separate two statements in a single line.)

The result would look like the following:



Example Sub Main()

```
'Creates a meter to plot sin(t). See also
'WMOutputColumn below.
Dim M as WMOutput
```

```
Set M = WM.ActiveDocument.NewOutput()  
M.Format = "graph"  
M.Column(1).Label = "sine plot"  
M.Column(1).Cell.Formula = "sin(t)"  
End Sub
```

See Also WMOutputColumn (object), WMDocument.NewOutput (method)

WMOutputColumn (object)

Syntax WMOutputColumn

Description An object which provides an interface to individual formulas of outputs (meters).

Comments A WMOutputColumn object represents one of the output fields of a WMOutput (meter) object. While a WMOutput object contains information regarding the meter display format, its WMOutputColumn objects hold information regarding what quantities are to be displayed in the meter.

Each WMOutputColumn object corresponds to a row in the Properties window for a meter. Just like in Working Model, you can specify the quantity to be measured (using formulas), minimum and maximum values for plotting (or automatic scale), and display the label for each output column.

The table below shows how the properties of the WMOutputColumn object can be specified.

Method/PropertyDescription

AutoScale	Boolean to indicate whether the particular output is subject to automatic scaling. AutoScale is True when the automatic scaling is active, and False otherwise.
Cell	WMCell containing the quantity to be measured. Typically, formulas are used to specify the quantity to be measured (see Example below).
Label	String which contains the label for the particular meter column. Label is no more than an informative comment to make the output easy to read.
Min	Double to specify the minimum bound of the plotting window when the WMOutput object is in graph format. When AutoScale is True, Min returns the current minimum bound.
Max	Double to specify the maximum bound of the plotting window when the WMOutput object is in graph format. When AutoScale is True, Max returns the current maximum bound.
Show	Boolean to indicate whether the column is to be displayed on the Output

object. Show is True when the value is displayed, or False otherwise.

Example

```
Sub Main()
    ' Creates a digital meter to show t and sin(t)
    Dim M as WMOutput
    Dim T1 as WMOutputColumn
    Set M = WM.ActiveDocument.NewOutput()
    M.Format = "digital"
    Set T1 = M.Column(2)
    T1.Label = "sine of t"
    T1.Cell.Formula = "sin(t)"
End Sub
```

See Also WMOutput (object)

WMPoint (object)

Syntax WMPoint

Description An object which provides an interface to points used in Working Model simulations.

Comments A WMPoint object has the following properties.

Note: WMPoint object also has properties available in WMObject objects. Please see the section on WMObject for details.

Property	Description
Kind	(String) indicates the type of the Point (point, squarepoint, or anchor).
PX, PY	(WMCell) x- and y-positions of the Point.
PR	(WMCell) orientation of the Point.
GlobalPX	(Double) global x-position of the Point.
GlobalPY	(Double) global y-position of the Point.
GlobalPR	(Double) global orientation of the Point.
Body	(WMBody) the body to which the Point is attached. If the point is attached to the background, the Body property returns the background, which has the ID of 0 (see Example).
Constraint	(WMConstraint) the constraint object which the Point is part of. Read-only. Returns Nothing if the point is not part of any constraint.

Example `Sub Main()
 ' Creates a square and attaches a point at a corner
 ' of the square.
 Dim D as WMDocument
 Dim P as WMPoint
 Dim B as WMBody
 Set D = WM.ActiveDocument
 ' Creates a square. By default, it will be located at (0, 0)
 Set B = D.NewBody("Square")
 B.Width.Value = 1.0 ' Height will also be 1.0
 ' Creates a point. By default, it will be located at (0, 0)
 Set P = D.NewPoint("Point")
 If P.Body.ID = 0 then
 MsgBox "The point is currently attached to the background."
 End If
 Set P.Body = B ' attached to the square previously created
 MsgBox "Now the point is attached to "+P.Body.Name
 ' Move the point to the corner of the square. Note Point's
 ' coordinates are given in terms of the body it is attached.
 P.PX.Value = 0.5 ' 0.5 in x from the FOR of the square
 P.PY.Value = 0.5 ' 0.5 in y from the FOR of the square
 End Sub`

See Also `WMCell (object), WMBody (object)`

Editing and Debugging Scripts

This chapter explains how to use Script Editor, a tool that enables you to edit and debug your WM Basic scripts. It begins with some general information about working with Script Editor and then discusses editing your scripts, running your scripts to make sure they work properly, debugging them if necessary, and exiting from Script Editor.

Contents

- Script Editor Basics
- Editing Your Scripts
- Running Your Scripts
- Debugging Your Scripts
- Exiting from Script Editor
- Menu Reference

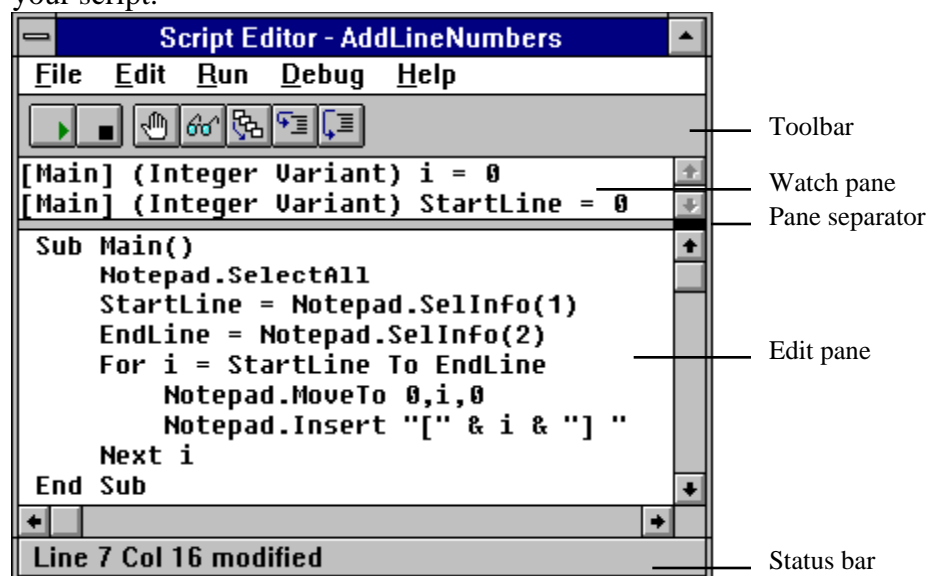
Script Editor Basics

This section provides general information that will help you work most effectively with Script Editor. It includes an overview of Script Editor's application window—the interface you'll use to edit, run, and debug your WM Basic scripts—as well as lists of keyboard shortcuts and information on using the Help system.

Script Editor's Application Window

Script Editor's application window contains the following elements:








- **Toolbar:** a collection of tools that you can use to provide instructions to Script Editor, as discussed in the following subsection.
- **Edit pane:** a window containing the WM Basic code for the script you are currently editing.
- **Watch pane:** a window that opens to display the watch variable list after you have added one or more variables to that list.
- **Pane separator:** a divider that appears between the edit pane and the watch pane when the watch pane is open.
- **Status bar:** displays the current location of the insertion point within your script.



Toolbar

The following list briefly explains the purpose of each of the tools on Script Editor's toolbar. These tools are discussed in more detail later in the chapter, in the context of the procedures in which they are used.

Toolbar Tools

Button Face	Tool	Function
	Start	Begins execution of a script.
	End	Stops execution of a script.
	Toggle Breakpoint	Adds or removes a breakpoint on a line of WM Basic code.
	Add Watch	Displays the Add Watch dialog box, in which you can specify the name of a WM Basic variable. That variable, together with its value (if any), is then displayed in the watch pane of Script Editor's application window.
	Calls	Displays the list of procedures called by the currently executing WM Basic script. Available only during break mode.
	Single Step	Executes the next line of a WM Basic script and then suspends execution of the script. If the script calls another WM Basic procedure, execution will continue into each line of the called procedure.
	Procedure Step	Executes the next line of a WM Basic script and then suspends execution of the script. If the script calls another WM Basic procedure, WM Basic will run the called procedure in its entirety.

Keyboard Shortcuts

The following lists present various types of keyboard shortcuts, including general shortcuts, navigating shortcuts, editing shortcuts, and debugging shortcuts.

General Shortcuts

Key(s)	Function
F1 (Windows only)	Provides context-sensitive help for selected menu commands and variables in the watch pane, for WM Basic terms in the edit pane that have been selected or that contain the insertion point, and for displayed dialog boxes.
Shift+F1 (Windows only)	Toggles the Help pointer.
Ctrl+F (Windows), Cmd+F (Macintosh)	Displays the Find dialog box, which allows you to specify text for which you want to search.
F3 (Windows), Cmd+G (Macintosh)	Searches for the next occurrence of previously specified text. If you have not previously specified text for which you want to search, displays the Find dialog box.
F4	Invokes the Goto Line dialog box.
Esc (Windows)	Deactivates the Help pointer if it is active. Otherwise, compiles your script and returns you to the host application.

Navigating Shortcuts

Key(s)	Function
Up arrow	Moves the insertion point up one line.
Down arrow	Moves the insertion point down one line.
Left arrow	Moves the insertion point left by one character position.
Right arrow	Moves the insertion point right by one character position.
PgUp	Moves the insertion point up by one window.
PgDn	Moves the insertion point down by one window.
Ctrl+PgUp (Windows)	Scrolls the insertion point left by one window.
Ctrl+PgDn (Windows)	Scrolls the insertion point right by one window.

Ctrl+ Left arrow (Windows), Cmd+Left arrow (Macintosh)	Moves the insertion point to the start of the next word to the left.
Ctrl+Right arrow (Windows), Cmd+Right arrow (Macintosh)	Moves the insertion point to the start of the next word to the right.
Home	On Windows, places the insertion point before the first character in the line. On Macintosh, scrolls back to the beginning of the script (without moving the cursor).
End	On Windows, places the insertion point after the last character in the line. On Macintosh, scrolls down to the end of the script (without moving the cursor).
Ctrl+Home (Windows)	Places the insertion point before the first character in the script.
Ctrl+End (Windows)	Places the insertion point after the last character in the script.

Editing Shortcuts

Key(s)	Function
Delete	Removes the selected text or removes the character following the insertion point without placing it on the Clipboard.
Backspace	Removes the selected text or removes the character preceding the insertion point without placing it on the Clipboard.
Ctrl+Y (Windows)	Deletes the entire line containing the insertion point without placing it on the Clipboard.
Tab	Inserts a tab character.
Enter	Inserts a new line, breaking the current line.
Ctrl+C (Windows), Cmd+C (Macintosh)	Copies the selected text, without removing it from the script, and places it on the Clipboard.

Ctrl+X (Windows), Cmd+X (Macintosh)	Removes the selected text from the script and places it on the Clipboard.
Ctrl+V (Windows), Cmd+V (Macintosh)	Inserts the contents of the Clipboard at the location of the insertion point.
Shift + any navigating shortcut	Selects the text between the initial location of the insertion point and the point to which the keyboard shortcut would normally move the insertion point. (For example, pressing Shift + Ctrl + Left arrow selects the word to the left of the insertion point.
Ctrl+Z (Windows), Cmd+Z (Macintosh)	Reverses the effect of the preceding editing change(s).

Debugging Shortcuts

Key(s)	Function
Shift+F9	Displays the Add Watch dialog box, in which you can specify the name of a WM Basic variable. Script Editor then displays the value of that variable, if any, in the watch pane of its application window.
Delete	Removes the selected watch variable from the watch pane.
Enter or F2	Displays the Modify Variable dialog box for the selected watch variable, which enables you to modify the value of that variable.
Cmd+Y (Macintosh)	Performs a syntax check on your script.
Cmd+T (Macintosh), F5	Runs your script.
Cmd+E (Macintosh)	Terminates your script.
F6	If the watch pane is open, switches the insertion point between the watch pane and the edit pane.

Cmd+="" (Macintosh), F8	Activates the Single Step command, which executes the next line of a WM Basic script and then suspends execution of the script. If the script calls another WM Basic procedure, execution will continue into each line of the called procedure.
Shift+F8	Activates the Procedure Step command, which executes the next line of a WM Basic script and then suspends execution of the script. If the script calls another WM Basic procedure, WM Basic will run the called procedure in its entirety.
Ctrl+Break (Windows) or Cmd+period (.) (Macintosh)	Suspends execution of an executing script and places the instruction pointer on the next line to be executed.
F9	Sets or removes a breakpoint on the line containing the insertion point.

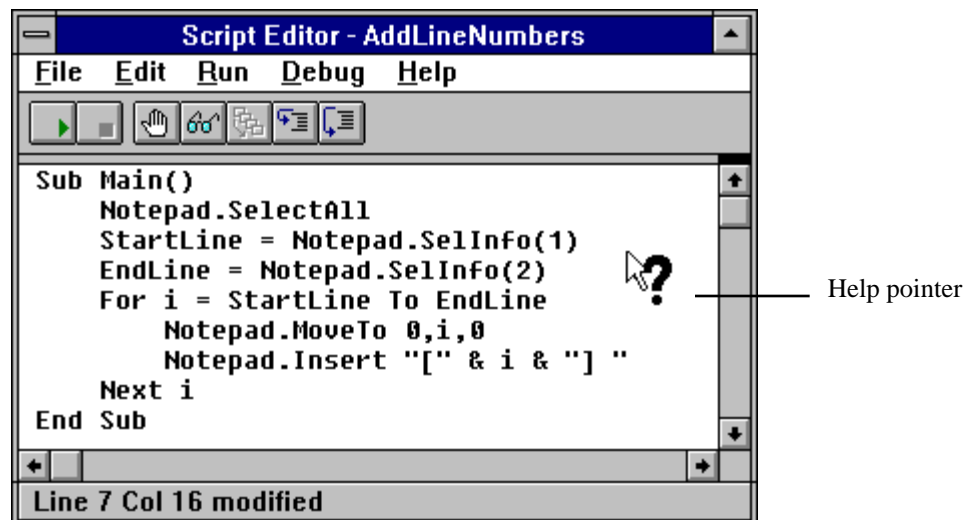
Using the Help System (Windows Only)

Script Editor's Help system provides context-sensitive help both on WM Basic keywords and on how to use various features of Script Editor. The Help system also lets you pinpoint information on key topics. This subsection describes several ways to get help.

Here's how to activate the Help pointer and use it to get help on certain key features of Script Editor, including the status bar, toolbar, menu commands, edit pane, watch pane, and pane separator.

To get context-sensitive help using the Help pointer:

1. To activate the Help pointer, press Shift+F1.
When you pass the pointer over an area of Script Editor's application window on which you can obtain help, a question mark appears beside the mouse pointer to indicate that the Help pointer is active.



Note: You can use the Help pointer to obtain help on *all* toolbar tools and menu commands, including those that are currently disabled.

2. Place the Help pointer on the item for which you want help and click the mouse button.
Script Editor's Help system displays information for the item on which you clicked.

Here's how to use the keyboard to get context-sensitive help on WM Basic terms, watch variables, menu commands, and dialog boxes.

To get context-sensitive help using the keyboard:

1. Select the WM Basic term on which you want help or place the insertion point anywhere in the term.
-Or-
Select the watch variable or menu command on which you want help (including commands that are currently disabled).
-Or-
Display the dialog box on which you want help.
2. Press F1.
Script Editor's Help system displays information on the WM Basic term, watch variable, menu command, or dialog box.

Here's how to access the Help system and pinpoint specific information within it.

To search for help on a specific topic:

1. From the Help menu, choose the Search for Help on command.
The Search dialog box appears.
2. Either enter the desired topic in the text box or select it from the following scrollable list.
3. Click the Show Topics button or press Enter.

The topic you selected is displayed in the second scrollable list, together with closely related Help topics, if any.

4. Click the Go To button or press Enter.

Help is displayed for the topic you selected.

Here's how to display the Help system contents and use it to find information on selected topics.

To use the Help system contents:

1. From the Help menu, choose Contents.

A list of major topics in Script Editor's Help system appears.

2. Select the topic on which you want help.

The Help system either displays information on the selected topic or presents a list of more specific subtopics from which you can choose to obtain the desired information.

Editing Your Scripts

This section explains how to use Script Editor to edit WM Basic code. Although, in some respects, editing code with Script Editor is like editing regular text with a word-processing program, Script Editor also has certain capabilities specifically designed to help you edit WM Basic code.

You'll learn how to move around within your script, select and edit text, add comments to your script, break long WM Basic statements across multiple lines, search for and replace selected text, and perform a syntax check of your script. The section ends with a brief discussion of editing dialog box templates, which is explained in much more detail in Chapter 3.

Navigating within a Script

The lists of keyboard shortcuts in the preceding section contain a group of navigating shortcuts, which you can use to move the insertion point around within your script. When you move the insertion point with a keyboard shortcut, Script Editor scrolls the new location of the insertion point into view if it is not already displayed.

You can also reposition the insertion point with the mouse and the Goto Line command, as explained below.

Script Editor differs from most word-processing programs in that it allows you to place the insertion point anywhere within your script, including in "empty spaces." (Empty spaces are areas within the script that do not contain text, such as a tab's expanded space or the area beyond the last character on a line.)

Here's how to use the mouse to reposition the insertion point. This approach is especially fast if the area of the screen to which you want to move the insertion point is currently visible.

To move the insertion point with the mouse:

1. Use the scroll bars at the right and bottom of the display to scroll the target area of the script into view if it is not already visible.
2. Place the mouse pointer where you want to position the insertion point.
3. Click the left mouse button.

The insertion point is repositioned.

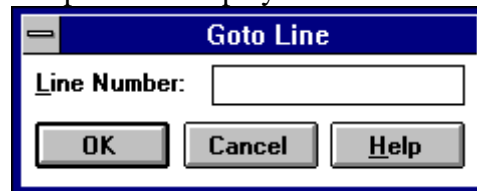
Note: When you scroll the display with the mouse, the insertion point remains in its original position until you reposition it with a mouse click. If you attempt to perform an editing operation when the insertion point is not in view, Script Editor automatically scrolls the insertion point into view before performing the operation.

Here's how to jump directly to a specified line in your script. This approach is especially fast if the area of the screen to which you want to move the insertion point is not currently visible but you know the number of the target line.

To move the insertion point to a specified line in your script:

1. Press F4.

Script Editor displays the Goto Line dialog box.



2. Enter the number of the line in your script to which you want to move the insertion point.

3. Click the OK button or press Enter.

The insertion point is positioned at the start of the line you specified. If that line was not already displayed, Script Editor scrolls it into view.

Note: The insertion point cannot be moved so far below the end of a script as to scroll the script entirely off the display. When the last line of your script becomes the first line on your screen, the script will stop scrolling, and you will be unable to move the insertion point below the bottom of that screen.

Performing Editing Operations with Script Editor

This subsection provides an overview of the editing operations you can perform with Script Editor—including inserting, selecting, deleting, cutting, copying, and pasting material—and explains how to undo, or reverse, the most recent editing operations. You may wish to refer to the lists of keyboard shortcuts in the preceding section, which contain a group of editing shortcuts that can be used to perform many of the operations discussed here.

Inserting Text

In Script Editor, inserting text and other characters such as tabs and line breaks works about the same way as it does in a word-processing program: you position the insertion point at the desired location in the script and start typing.

However, as noted in the preceding subsection, Script Editor lets you position the insertion point in "empty spaces," which means that you can also insert text into empty spaces—a feature that comes in handy when you want to insert a comment in the space beyond the end of a line in your script. (Adding comments to your script is discussed later in this section.) When you insert characters beyond the end of a line, the space between the insertion point and the last character on the line is backfilled with tab characters.

Another way in which Script Editor differs from word-processing programs is that in Script Editor, text does not wrap. If you keep entering text on a given line, eventually you will reach a point at which you can enter no more text on that line. Therefore, *you* control the line breaks by pressing Enter when you want to insert a new line in your script. The effect of pressing Enter depends on where the insertion point is located at the time:

- If you press Enter with the insertion point at or beyond the end of a line, a new line is inserted after the current line.
- If you press Enter with the insertion point at the start of a line, a new line is inserted before the current line.
- If you press Enter with the insertion point within a line, the current line is broken into two lines at that location.

If you press Tab, a tab character is inserted at the location of the insertion point, which causes text after the tab to be moved to the next tab position. If you insert new text within a tab's expanded space, the text that originally appeared on that line is moved to the next tab position each time the new text that you are entering reaches the start of another tab position.

Selecting Text

You can use either the mouse or the keyboard to select text and other characters in your script. Regardless of which method you use, you should be aware that in Script Editor, you can select either a portion of one line or a series of whole lines, but you cannot select a portion of one line plus one or more whole lines. When you are selecting multiple lines and start or end your selection partway through a line, Script Editor automatically extends the selection to include the entire starting and ending lines.

Here's how to use the mouse to select text in your script.

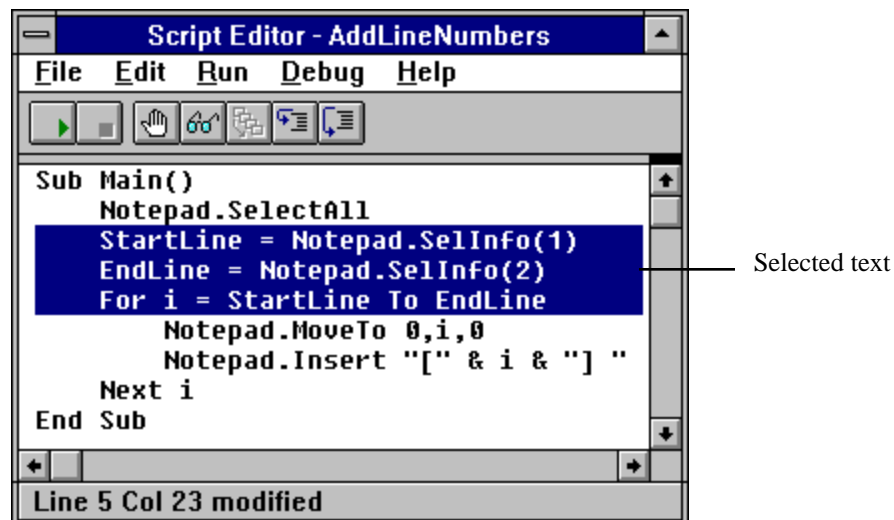
To select text with the mouse:

1. Place the mouse pointer where you want your selection to begin.
2. While pressing the left mouse button, drag the mouse until you reach the end of your selection, and release the mouse button.

-Or-

While pressing Shift, place the mouse pointer where you want your selection to end and click the left mouse button.

The selected text is highlighted on your display.



Another way to select one or more whole lines with the mouse is to start by placing the mouse pointer in the left margin beside the first line you want to select. The pointer becomes a reverse arrow, which points toward the line of text. Click the left mouse button to select a single line; press the left mouse button and drag up or down to select multiple lines. Here's how to use keyboard shortcuts to select text in your script.

To select text with the keyboard:

1. Place the insertion point where you want your selection to begin.
 2. While pressing Shift, use one of the navigating keyboard shortcuts to extend the selection to the desired ending point.
- The selected text is highlighted on your display.

Note: When you intend to select an entire single line of text in your script, it is important to remember to extend your selection far enough to include the hidden end-of-line character, which is the character that inserts a new line in your script.

Here's how to use the keyboard to select one or more whole lines in your script.

To select an entire line of text with the keyboard:

1. Place the insertion point at the beginning of the line you want to select.
2. Press Shift + Down arrow.
The entire line, including the end-of-line character, is selected.
3. To extend your selection to include additional whole lines of text, repeat step 2.

Once you have selected text within your script, you can perform a variety of other editing operations on it, including deleting the text, placing it on the Clipboard (either by cutting the text or copying it), and pasting it.

Deleting Text

When you delete material, it is removed from your script without being placed on the Clipboard.

Here's how to remove one or more characters, selected text, or entire lines from your script.

To delete text:

- To remove a single character to the left of the insertion point, press Backspace once; to remove a single character to the right of the insertion point, press Delete once. To remove multiple characters, hold down Backspace or Delete.

-Or-

To remove text that you have selected, press Backspace or Delete.

-Or-

(On Windows) To remove an entire line, place the insertion point in that line and press Ctrl+Y.

Here's how to remove an unwanted line break from your script.

To combine the current line with the following line:

1. Place the insertion point after the last character on the current line.
2. Press Delete once to delete the hidden end-of-line character.
The current line and the following line are combined.

Notes: If any spaces were entered at the end of the current line, you may have to press Delete one or more additional times to remove these hidden characters first before you can delete the end-of-line character. Pressing Backspace with the insertion point at the start of a line has no effect—that is, it will not combine the current line with the preceding line.

Cutting and Copying Text

You can place material from your script on the Clipboard by either cutting it or copying it. Here's how to place on the Clipboard text that you have cut from your script.

To cut a selection:

- Press Ctrl+X (Windows) or Cmd+X (Macintosh).
The selection is removed from your script and placed on the Clipboard.

Here's how to place on the Clipboard text that you have copied from your script.

To copy a selection:

- Press Ctrl+C (Windows) or Cmd+C (Macintosh).
The selection remains in your script, and a copy of it is placed on the Clipboard.

Pasting Text

Once you have cut or copied material to the Clipboard, here's how to paste it into your script at another location.

To paste the contents of the Clipboard into your script:

1. Position the insertion point where you want to place the contents of the Clipboard.

2. Press Ctrl+V (Windows) or Cmd+V (Macintosh).
The contents of the Clipboard appear at the location of the insertion point.

If you wish to delete a block of text and insert the contents of the Clipboard in its place, you can combine the two operations by first selecting the text you want to remove and then pressing Ctrl+V (Windows) or Cmd+V (Macintosh) to replace it with the contents of the Clipboard.

Undoing Editing Operations

You can undo editing operations that produce a change in your script, including:

- The insertion of a series of characters
- The insertion of a block of text from the Clipboard
- The deletion of a series of characters
- The deletion or cutting of a block of text

You cannot undo operations that don't produce any change in your script, such as moving the insertion point, selecting text, and copying material to the Clipboard.

Here's how to reverse the effect of the preceding editing operation.

To undo an editing operation:

- Press Ctrl+Z (Windows) or Cmd+Z (Macintosh).
Your script is restored to the way it looked before you performed the editing operation.

Adding Comments to Your Script

You can add comments to your script to remind yourself or others of how your code works. Comments are ignored when your script is executed.

In WM Basic, the apostrophe symbol (') is used to indicate that the text from the apostrophe to the end of the line is a comment.

Here's how to designate an entire line as a comment.

To add a full-line comment:

1. Type an apostrophe (') at the start of the line.
2. Type your comment following the apostrophe.
When your script is run, the presence of the apostrophe at the start of the line will cause the entire line to be ignored.

Here's how to designate the last part of a line as a comment.

To add a comment at the end of a line of code:

1. Position the insertion point in the empty space beyond the end of the line of code.
2. Type an apostrophe (').
3. Type your comment following the apostrophe.
When your script is run, the code on the first portion of the line will be executed, but the presence of the apostrophe at the start of the comment will cause the remainder of the line to be ignored.

Although you can place a comment at the end of a line containing executable code, you cannot place executable code at the end of a line containing a comment because the presence

of the apostrophe at the start of the comment will cause the balance of the line (including the code) to be ignored.

Breaking a WM Basic Statement across Multiple Lines

By default, in Script Editor, a single WM Basic statement can extend only as far as the right margin, and each line break represents a new statement. However, you can override this default if you want to break a long statement across two or more lines.

Here's how to indicate that two or more lines of WM Basic code should be treated as a single statement when your script is run.

To break a WM Basic statement across multiple lines:

1. Type the WM Basic statement on multiple lines, exactly the way you want it to appear.
2. Place the insertion point at the end of the first line in the series.
3. Press the spacebar once to insert a single space.
4. Type an underscore (_).

Note: The underscore is the *line-continuation character*, which indicates that the WM Basic statement continues on the following line.

5. Repeat steps 2–4 to place a line-continuation character at the end of each line in the series except the last.
When you run your script, the code on this series of lines will be executed as a single WM Basic statement, just as if you had typed the entire statement on the same line.

Searching and Replacing

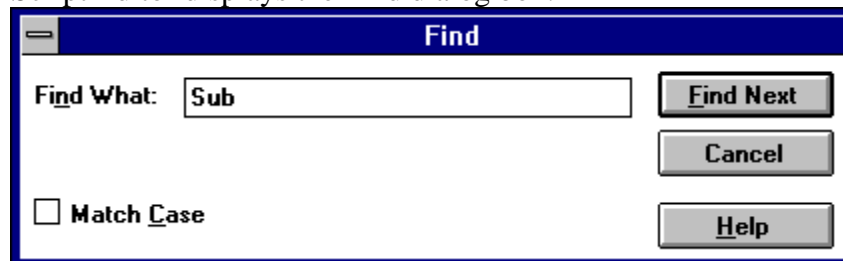
Script Editor makes it easy to search for specified text in your script and automatically replace instances of specified text.

Finding Text in Your Script

Here's how to locate instances of specified text quickly anywhere within your script.

To find specified text:

1. Move the insertion point to where you want to start your search. (To start at the beginning of your script, press Ctrl+Home (Windows) or Home (Macintosh).)
2. Press Ctrl+F (Windows) or Cmd+F (Macintosh).
Script Editor displays the Find dialog box:



3. In the Find What field, specify the text you want to find.

4. Select the Match Case check box if you want the search to be case-sensitive. Otherwise, the search will be case-insensitive.
5. Click the Find Next button or press Enter.
The Find dialog box remains displayed, and Script Editor either highlights the first instance of the specified text or indicates that it cannot be found.
6. If the specified text has been found, repeat step 5 to search for the next instance of it.

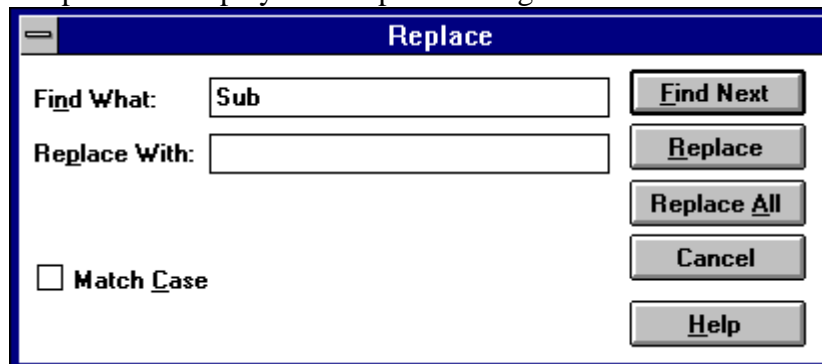
Note: If the Find dialog box blocks your view of an instance of the specified text, you can move the dialog box out of your way and continue with your search. You can also click the Cancel button, which removes the Find dialog box while maintaining the established search criteria, and then press F3 to find successive occurrences of the specified text. (If you press F3 when you have not previously specified text for which you want to search, Script Editor displays the Find dialog box so you can specify the desired text.)

Replacing Text in Your Script

Here's how you can automatically replace either all instances or selected instances of specified text.

To replace specified text:

1. Move the insertion point to where you want to start the replacement operation. (To start at the beginning of your script, press Ctrl+Home on Windows or Home on Macintosh.)
2. Choose the Replace command from the Search menu.
Script Editor displays the Replace dialog box:



3. In the Find What field, specify the text you want to replace.
4. In the Replace With field, specify the replacement text.
5. Select the Match Case check box if you want the replacement operation to be case-sensitive. Otherwise, it will be case-insensitive.
6. To replace all instances of the specified text, click the Replace All button.

Script Editor either replaces the specified text throughout your script and indicates the number of occurrences it has changed, or it indicates that the specified text cannot be found.

7. To replace selected instances of the specified text, click the Find Next button.

Script Editor either highlights the first instance of the specified text or indicates that it cannot be found.

8. If the specified text has been found, either click the Replace button to replace that instance of it or click the Find Next button to highlight the next instance (if any).

Each time you click the Replace button, Script Editor replaces that instance of the specified text and automatically highlights the next instance.

Checking the Syntax of a Script

When you try to run or debug a script whose syntax hasn't been checked, Script Editor first performs a syntax check automatically.

Here's how to perform a syntax check manually when you are editing your script, without having to run it.

To perform a syntax check:

1. From the Run menu, choose the Syntax Check command.
Script Editor either indicates that no errors have been found or displays an error message that specifies the first line in your script where an error has been found and briefly describes the nature of that error.
2. Click the OK button or press Enter.
If Script Editor has found a syntax error, the line containing the error is highlighted on your display.
3. Correct the syntax error.
4. Repeat steps 1–3 until you have found and corrected all syntax errors.

Editing Dialog Box Templates (Windows Only)

If you are running the WM Basic on Windows, the Insert New Dialog and Edit Dialog commands will appear in the Edit menu. These commands allow you to use the features of Dialog Editor to create a new dialog box template and insert it into your script or edit an existing dialog box template contained in your script.

Here's how to invoke Dialog Editor and use it to create a new dialog box template for use in your script.

To insert a new dialog box template into your script:

1. Place the insertion point where you want the new dialog box template to appear in your script.
2. From the Edit menu, choose the Insert New Dialog command.
Script Editor's application window is temporarily disabled, and Dialog Editor appears, displaying a new dialog box in its application window.
3. Use Dialog Editor to create your dialog box.

4. Exit from Dialog Editor and return to Script Editor.
Script Editor automatically places the new dialog box template generated by Dialog Editor in your script at the location of the insertion point.

Here's how to invoke Dialog Editor and use it to modify a dialog box template contained in your script.

To edit an existing dialog box template in your script:

1. Select the WM Basic code for the entire dialog box template.
2. From the Edit menu, choose the Edit Dialog command.
Script Editor's application window is temporarily disabled, and Dialog Editor appears, displaying in its application window a dialog box created from the code you selected.
3. Use Dialog Editor to modify your dialog box.
4. Exit from Dialog Editor and return to Script Editor.
Script Editor automatically replaces the dialog box template you originally selected with the revised template generated by Dialog Editor.

Refer to Chapter 4 for a detailed discussion of how to use Dialog Editor to create and edit dialog box templates.

Running Your Scripts

Once you have finished editing your script, you will want to run it to make sure it performs the way you intended. You can also pause or stop an executing script.

Here's how to compile your script, if necessary, and then execute it.

To run your script:

- Click the Start tool on the toolbar.

-Or-

Press F5.

The script is compiled (if it has not already been compiled), the focus is switched to the Working Model window, and the script is executed.

Note: During script execution, Script Editor's application window is available only in a limited manner. Some of the menu commands may be disabled, and the toolbar tools may be inoperative.

Here's how to suspend the execution of a script that you are running.

To pause an executing script:

- Press Ctrl+Break (Windows) or Cmd+ "." (period) (Macintosh).
Execution of the script is suspended, and the instruction pointer (a gray highlight) appears on the line of code where the script stopped executing.

Note: The instruction pointer designates the line of code that will be executed next if you resume running your script.

Here's how to stop the execution of a script that you are running.

To stop an executing script:

- Click the End tool on the toolbar.

Note: Many of the functions of Script Editor's application window may be unavailable while you are running a script. If you want to stop your script but find that the toolbar is currently inoperative, press Ctrl+C (Windows) or Cmd+"." (period) (Macintosh) to pause your script, then click the End tool.

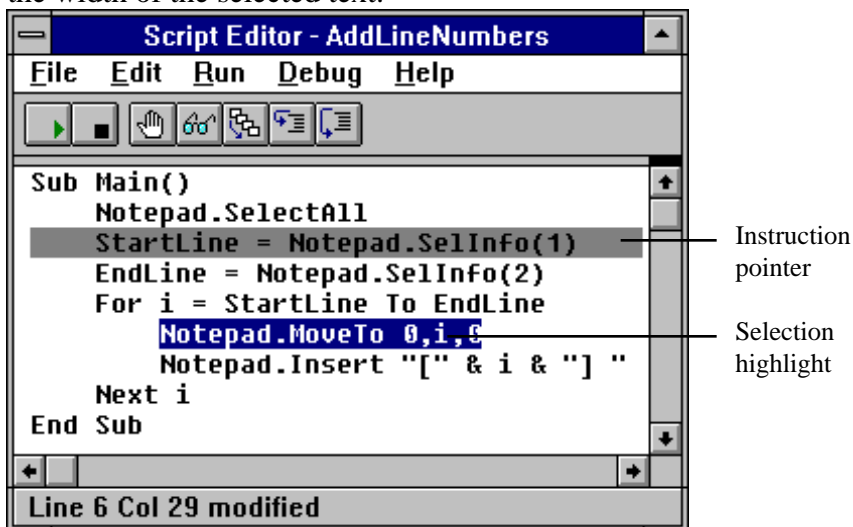
Debugging Your Scripts

This section presents some general information that will help you work most effectively with Script Editor's debugging capabilities and then explains how to trace the execution of your script, how to set and remove breakpoints, and how to add watch variables and modify their value.

Using the WM Basic Debugger

While debugging, you are actually executing the code in your script line by line. Therefore, to prevent any modifications to your script while it is being run, the edit pane is read-only during the debugging process. You are free to move the insertion point throughout the script, select text and copy it to the Clipboard as necessary, set breakpoints, and add and remove watch variables, but you cannot make any changes to the script until you stop running it.

To let you follow and control the debugging process, Script Editor displays an *instruction pointer* on the line of code that is about to be executed—that is, the line that will be executed next if you either proceed with the debugging process or run your script at full speed. When the instruction pointer is on a line of code, the text on that line appears in black on a gray background that spans the width of the entire line. The following illustration shows the difference between the instruction pointer and the selection highlight (discussed in the preceding section), in which the text appears in white on a black background that spans only the width of the selected text.



Tracing Script Execution

Script Editor gives you two ways to trace script execution—single step and procedure step—both of which involve stepping through your script code line by line. The distinction between the two is that the single step process traces into calls to user-defined functions and subroutines, whereas the procedure step process does not trace into these calls (although it does execute them).

Here's how to trace the execution of your script with either the single step or procedure step method.

To step through your script:

1. Click the Single Step or Procedure Step tool on the toolbar.
-Or-
Press F8 (Single Step) or Shift+F8 (Procedure Step).
Script Editor places the instruction pointer on the sub main line of your script.

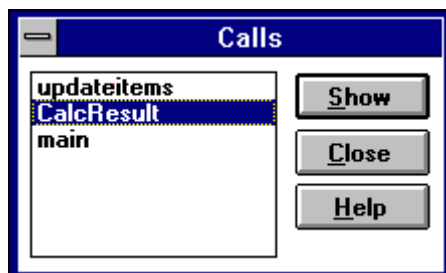
Note: When you initiate execution of your script with any of these methods, the script will first be compiled, if necessary. Therefore, there may be a slight pause before execution actually begins. If your script contains any compile errors, it will not be executed. To debug your script, first correct any compile errors, then initiate execution again.

2. To continue tracing the execution of your script line by line, repeat step 1.
Each time you repeat step 1, Script Editor executes the line containing the instruction pointer and moves the instruction pointer to the next line to be executed.
3. When you finish tracing the execution of your script, either click the Start tool on the toolbar (or press F5) to run the balance of the script at full speed or click the End tool to halt execution of the script.

When you are stepping through a subroutine, you may need to determine the procedure calls by which you arrived at that point in your script. Here's how to use the Calls dialog box to obtain this information.

To display the Calls dialog box:

1. Click the Calls tool on the toolbar.
Script Editor displays the Calls dialog box, which lists the procedure calls made by your script in the course of arriving at the present subroutine.



2. From the Calls dialog box, select the name of the procedure you wish to view.
3. Click the Show button.
Script Editor highlights the currently executing line in the procedure you selected, scrolling that line into view if necessary. (During this process, the instruction pointer remains in its original location in the subroutine.)

When you are stepping through a subroutine, you may want to repeat or skip execution of a section of code. Here's how to use the Set Next Statement command to move the instruction pointer to another line within that subroutine.

To move the instruction pointer to another line within a subroutine:

1. Place the insertion point in the line where you want to resume stepping through the script.
2. From the Debug menu, choose the Set Next Statement command.
The instruction pointer moves to the line you selected, and you can resume stepping through your script from there.

Note: You can only use the Set Next Statement command to move the instruction pointer within the same subroutine. If you place the insertion point on a line that is not in the same subroutine, the Set Next Statement command will be disabled in the Debug menu.

Setting and Removing Breakpoints

If you want to start the debugging process at the first line of your script and then step through your code line by line until you reach the end of the code that you need to debug, the method described in the preceding subsection works fine. But if you only need to debug one or more portions of a long script, that method can be pretty cumbersome.

An alternate strategy is to set one or more *breakpoints* at selected lines in your script. Script Editor suspends execution of your script just before it reaches a line containing a breakpoint, thereby allowing you to begin or resume stepping through the script from that point.

Setting Breakpoints

You can set breakpoints to begin the debugging process partway through your script, to continue debugging at a line outside the current subroutine, and to debug only selected portions of your script.

Valid breakpoints can only be set on lines in your script that contain code, including lines in functions and subroutines. Although you can set a breakpoint anywhere within a script prior to execution, when you compile and run the script, invalid breakpoints (that is, breakpoints

on lines that don't contain code) are automatically removed. While you are debugging your script, Script Editor will beep if you try to set a breakpoint on a line that does not contain code.

Here's how to begin the debugging process at a selected point in your script.

To start debugging partway through a script:

1. Place the insertion point in the line where you want to start debugging.
2. To set a breakpoint on that line, click the Toggle Breakpoint tool on the toolbar.

-Or-

Press F9.

The line on which you set the breakpoint now appears in contrasting type.

3. Click the Start tool on the toolbar.

-Or-

Press F5.

Script Editor runs your script at full speed from the beginning and then pauses prior to executing the line containing the breakpoint. It places the instruction pointer on that line to designate it as the line that will be executed next when you either proceed with debugging or resume running the script.

If you want to continue debugging at another line in your script, you can use the Set Next Statement command in the Debug menu to move the instruction pointer to the desired line—provided the line is within the same subroutine.

If you want to continue debugging at a line that isn't within the same subroutine, here's how to move the instruction pointer to that line.

To continue debugging at a line outside the current subroutine:

1. Place the insertion point in the line where you want to continue debugging.
2. To set a breakpoint on that line, press F9.
3. To run your script, click the Start tool on the toolbar or press F5.
The script executes at full speed until it reaches the line containing the breakpoint and then pauses with the instruction pointer on that line.

You can now resume stepping through your script from that point.

If you only need to debug parts of your script, here's how to facilitate the task by using breakpoints.

To debug selected portions of your script:

1. Place a breakpoint at the start of each portion of your script that you want to debug.

Note: Up to 255 lines in your script can contain breakpoints.

2. To run the script, click the Start tool on the toolbar or press F5.
The script executes at full speed until it reaches the line containing the first breakpoint and then pauses with the instruction pointer on that line.
3. Step through as much of the code as you need to.

4. To resume running your script, click the Start tool on the toolbar or press F5.
The script executes at full speed until it reaches the line containing the second breakpoint and then pauses with the instruction pointer on that line.
5. Repeat steps 3 and 4 until you have finished debugging the selected portions of your script.

Removing Breakpoints

Breakpoints can be removed either manually or automatically. Here's how to delete breakpoints manually one at a time.

To remove a single breakpoint manually:

1. Place the insertion point on the line containing the breakpoint that you want to remove.
2. Click the Toggle Breakpoint tool on the toolbar.
-Or-
Press F9.
The breakpoint is removed, and the line no longer appears in contrasting type.

Here's how to delete all breakpoints manually in a single operation.

To remove all breakpoints manually:

- Select the Clear All Breakpoints command from the Debug menu.
Script Editor removes all breakpoints from your script.

Breakpoints are removed automatically under the following circumstances: (1) As mentioned earlier, when your script is compiled and executed, breakpoints are removed from lines that don't contain code. (2) When you exit from Script Editor, all breakpoints are cleared.

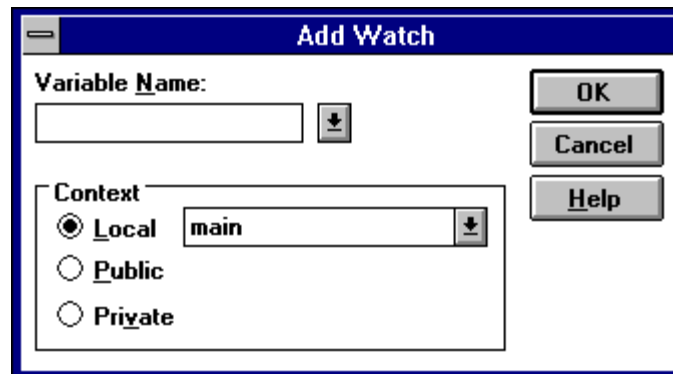
Adding a Watch Variable

As you debug your script, you can use Script Editor's watch pane to monitor selected variables. For each of the variables on this watch variable list, Script Editor displays the name of the variable, where it is defined, its value (if the variable is not in scope, its value is shown as <not in context>), and other key information such as its type and length (if it is a string). The values of the variables on the watch list are updated each time you enter break mode.

Here's how to add a variable to Script Editor's watch variable list.

To add a watch variable:

1. Click the Add Watch tool on the toolbar.
-Or-
Press Shift+F9.
Script Editor displays the Add Watch dialog box.



2. Use the controls in the Context box to specify where the variable is defined (locally, publicly, or privately) and, if it is defined locally, in which routine it is defined.
3. In the Variable Name field, enter the name of the variable you want to add to the watch variable list.

You can only watch variables of fundamental data types, such as Integer, Long, Variant, and so on; you cannot watch complex variables such as structures, arrays, or Working Model objects. You can, however, watch individual elements of arrays or structure members using the following syntax:

`[variable [(index, ...)] [.member [(index, ...)]] ...]`

Where *variable* is the name of the structure or array variable, *index* is a literal number, and *member* is the name of a structure member.

For example, the following are valid watch expressions:

Watch Variable	Description
<code>a(1)</code>	Element 1 of array <code>a</code>
<code>person.age</code>	Member <code>age</code> of structure <code>person</code>
<code>company(10,23).person.age</code>	Member <code>age</code> of structure <code>person</code> that is at element 10,23 within the array of structures called <code>company</code>

Note: If you are executing the script, you can display the names of all the variables that are "in scope," or defined within the current function or subroutine, on the drop-down Variable Name list and select the variable you want from that list.

4. Click the OK button or press Enter.
If this is the first variable you are placing on the watch variable list, the watch pane opens far enough to display that variable. If the watch pane was already open, it expands far enough to display the variable you just added.

Note: Although you can add as many watch variables to the list as you want, the watch pane only expands until it fills half of Script Editor's application window. If your list of watch variables becomes longer

than that, you can use the watch pane's scroll bars to bring hidden portions of the list into view.

The list of watch variables is maintained between script executions. Depending on the implementation, it may also be maintained between invocations of Script Editor. In order to delete a variable from Script Editor's watch variable list or modify the value of a variable on the list, you must first select the desired variable. Here's how to select a variable on the list.

To select a watch variable:

- Place the mouse pointer on the variable you want to select and click the left mouse button.

-Or-

If one of the variables on the watch list is already selected, use the arrow keys to move the selection highlight to the desired variable.

-Or-

If the insertion point is in the edit pane, press F6 to highlight the most recently selected variable on the watch list and then use the arrow keys to move the selection highlight to the desired variable.

Note: Pressing F6 again returns the insertion point to its previous position in the edit pane.

Here's how to delete a selected variable from Script Editor's watch variable list.

To delete a watch variable:

1. Select the variable on the watch list.
2. Press Delete.

The selected variable is removed from the watch list.

Modifying the Value of a Variable

When the debugger has control, you can modify the value of any of the variables on Script Editor's watch variable list. Here's how to change the value of a selected watch variable.

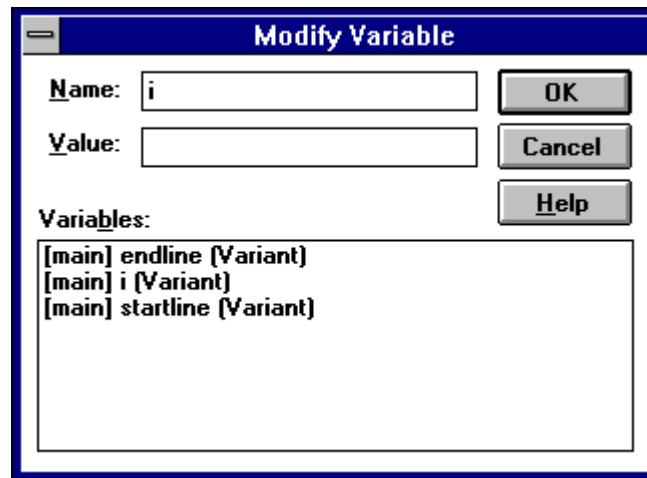
To modify the value of a variable on the watch variable list:

1. Place the mouse pointer on the name of the variable whose value you want to modify and double-click the left mouse button.

-Or-

Select the name of the variable whose value you want to modify and press Enter or F2.

Script Editor displays the Modify Variable dialog box.



Notes: The name of the variable you selected on the watch variable list appears in the Name field. If you want to change another variable, you can either enter a different variable in the Name field or select a different variable from the Variables list box, which shows the names of the variables that are defined within the current function or subroutine.

When you use the Modify Variable dialog box to change the value of a variable, you don't have to specify the context. Script Editor first searches locally for the definition of that variable, then privately, then publicly.

2. Enter the new value for your variable in the Value field.
3. Click the OK button.

The new value of your variable appears on the watch variable list.

When changing the value of a variable, Script Editor converts the new value to be of the same type as the variable being changed. For example, if you change the value of an Integer variable to 1.7, a conversion between a floating-point number and an Integer is performed, assigning the value 2 to the Integer variable.

When modifying a Variant variable, Script Editor needs to determine both the type and value of the data. Script Editor uses the following logic in performing this assignment (in this order):

If the new value is	Then
Null	The Variant variable is assigned Null (VarType 1)
Empty	The Variant variable is assigned Empty (VarType 0).
True	The Variant variable is assigned True (VarType 11).
False	The Variant variable is assigned False (VarType 11).

<i>number</i>	The <code>Variant</code> variable is assigned the value of <i>number</i> . The type of the variant is the smallest data type that fully represents that number. You can force the data type of the variable using a type-declarator letter following <i>number</i> , such as %, #, &, !, or @.
<i>date</i>	The <code>Variant</code> variable is assigned the value of the new date (<code>VarType 7</code>)
Anything else	The <code>Variant</code> variable is assigned a <code>String</code> (<code>VarType 8</code>).

Script Editor will not assign a new value if it cannot be converted to the same type as the specified variable.

Exiting from Script Editor

Here's how to get out of Script Editor. What happens when you exit depends on (1) whether you have made changes to your script and (2) whether your script contains errors.

To exit from Script Editor:

- Choose the Exit and Return command from the File menu.
If you *have* made changes to your script, Script Editor displays a dialog box asking whether you want to save the script. If you either click the No button or click the Yes button and your script contains no errors, you exit from Script Editor immediately. If you click the Yes button and your script contains errors, Script Editor highlights the line containing the first error and displays a dialog box asking whether you want to exit anyway. If you click the Yes button, Script Editor saves your script, errors and all, and then you exit from Script Editor.
If you *haven't* made any changes to your script, you exit from Script Editor immediately, regardless of whether the script contains errors from a previous editing session.

Menu Reference

File Menu

Command	Keyboard Shortcut	Function
New (Windows) New Script (Macintosh)	Ctrl+N (Windows) Cmd+N (Macintosh)	Creates a new script document.

<u>O</u> pen (Windows)	Ctrl+O (Windows)	Opens an existing script document.
Open Script (Macintosh)	Cmd+O (Macintosh)	
C <u>l</u> ose (Macintosh Only)	Cmd+W	Closes the Script Editor window and returns you to Working Model.
<u>S</u> ave	Cmd+S (Macintosh)	Saves the current script document under its filename.
Save <u>A</u> s		Saves the current script document under a new filename.
New Working Model Document (Macintosh)		Creates a new Working Model document.
Open Working Model Document (Macintosh)		Open an existing Working Model document.
Quit Working Model (Macintosh)	Cmd+Q	Closes Script Editor and quits Working Model.
<u>E</u> xit (Windows)	Alt+F4 or Ctrl+W	Closes the current script document and returns you to Working Model.

Edit Menu

Comman d	Keyboard Shortcut	Function
<u>U</u> ndo	Ctrl+Z (Windows), Cmd+Z (Mantic)	Reverses the effect of the preceding editing change(s).
C <u>u</u> t	Ctrl+X (Windows), Cmd+X (Macintosh)	Removes the selected text from the script and places it on the Clipboard.

<u>C</u> opy	Ctrl+C (Windows), Cmd+C (Macintosh)	Copies the selected text, without removing it from the script, and places it on the Clipboard.
<u>P</u> aste	Ctrl+V (Windows), Cmd+V (Macintosh)	Inserts the contents of the Clipboard at the current position of the insertion point.
<u>D</u> elete	Delete or Backspace	Removes the selected text from the script without placing it on the Clipboard.
<u>I</u> nsert New Dialog... (Windows)		(Windows Only) Invokes the Dialog Editor, which you can use to create a new dialog box for insertion into your script.
<u>E</u> dit Dialog... (Windows)		(Windows Only) Invokes the Dialog Editor, which you can use to edit the selected dialog box template. (This command is only enabled if a dialog box template is currently selected.)
<u>F</u> ind...	Ctrl+F (Windows), Cmd+F (Macintosh)	Displays the Find dialog box, which allows you to specify text for which you want to search.
Find <u>N</u> ext	F3 (Windows), Cmd+G (Macintosh)	Searches for the next occurrence of previously specified text. If you have not previously specified text for which you want to search, displays the Find dialog box.
<u>R</u> eplace...	Cmd+R (Macintosh)	Displays the Replace dialog box, which allows you to substitute replacement text for instances of specified text.
<u>G</u> oto Line...	F4	Presents the Goto Line dialog box, which allows you to move the insertion point to the start of a specified line number in your script.

Note: The Insert New Dialog and Edit Dialog commands only appear in the Edit menu if you are running the WM Basic 2.1 on a platform that supports Dialog Editor.

Run Menu

Command	Keyboard Shortcut	Function
<u>S</u> tart	Cmd+T (Macintosh), F5	Begins execution of a script.
<u>E</u> nd	Cmd+E (Macintosh)	Stops execution of an executing script.
Syntax C <u>h</u> eck	Cmd+Y (Macintosh)	Verifies the syntax of the statements in your script by compiling it.

Debug Menu

Command	Keyboard Shortcut	Function
<u>A</u> dd W <u>a</u> ch...	Shift+F9	Displays the Add Watch dialog box, in which you can specify the name of a WM Basic variable. That variable, together with its value (if any), is then displayed in the watch pane of Script Editor's application window.
<u>D</u> elete W <u>a</u> ch	Delete	Deletes a selected variable from the watch variable list.
<u>M</u> odify...	Enter or F2	Displays the Modify Variable dialog box for a selected variable, which enables you to modify the value of that variable.
<u>S</u> ingle Step	Cmd+="" (Macintosh), F8	Steps through the script code line by line, tracing into called procedures.
<u>P</u> rocedure S <u>t</u> ep	Shift+F8	Steps through the script code line by line without tracing into called procedures.
<u>T</u> oggle B <u>r</u> eakpoint	Cmd+B (Macintosh), F9	Toggles a breakpoint on the line containing the insertion point.

<u>C</u> lear All Breakpoints	Removes all breakpoints previously set with the Toggle Breakpoint command.
Set <u>N</u> ext Statement	Enables you to place the instruction pointer on another line within the current procedure and resume script execution from that point.

Help Menu (Windows Only)

Comman d	Keyboard Shortcut	Function
<u>C</u> ontents		Displays a list of major topics on which you can obtain help.
<u>S</u> earch for Help on...		Displays the Search dialog box, which allows you to search for Help topics containing specific keywords.

Editing Custom Dialog Boxes (Windows Only)

Dialog Editor is a tool that enables you to create and modify custom dialog boxes for use in your WM Basic scripts. Although the WM Basic statements used to display a dialog box and respond to the choices made by a user of the dialog box may seem complicated, Dialog Editor makes it easy to generate the WM Basic statements needed for your custom dialog boxes.

Note: Currently, Dialog Editor is available only on Working Model for Windows. However, Working Model for Macintosh is perfectly capable of running scripts that include dialogs. For sample code, see entries in Chapter 2 that are related to Dialog Manipulation, Predefined Dialogs, and User Dialogs (the list of such entries can be found in section “Language Elements by Category” of the Introduction).

Contents

- Overview
- Using the Dialog Editor
- Creating a Custom Dialog Box
- Editing a Custom Dialog Box
- Editing an Existing Dialog Box
- Testing an Edited Dialog Box
- Incorporating a Dialog Box Template into Your Script
- Exiting from Dialog Editor
- Menu Reference

Overview

Sometimes your script will need to obtain information from the user. In many cases, you can obtain this information by using one of WM Basic's predefined dialog boxes in your script. When you must go beyond the information-gathering capabilities provided by predefined dialog boxes, you can use Dialog Editor to create a custom dialog box for use in your script.

Dialog Editor is a tool that allows you to generate a *dialog box template* in WM Basic simply by editing an on-screen dialog box layout. You can then incorporate the template that Dialog Editor generates into your script.

The balance of this section provides general information that you'll need in order to work with Dialog Editor, including:

- Features that Dialog Editor supports
- An introduction to Dialog Editor's application window
- A list of keyboard shortcuts
- How to use the Help system

Then, in the following sections, you'll learn how to use Dialog Editor to create and edit custom dialog boxes and to edit dialog boxes captured from other applications. You'll also learn how to test an edited dialog box and incorporate the dialog box template generated by Dialog Editor into your script. And finally, you'll learn how to exit from Dialog Editor.

Features of the Dialog Editor

Dialog Editor supports the following features:

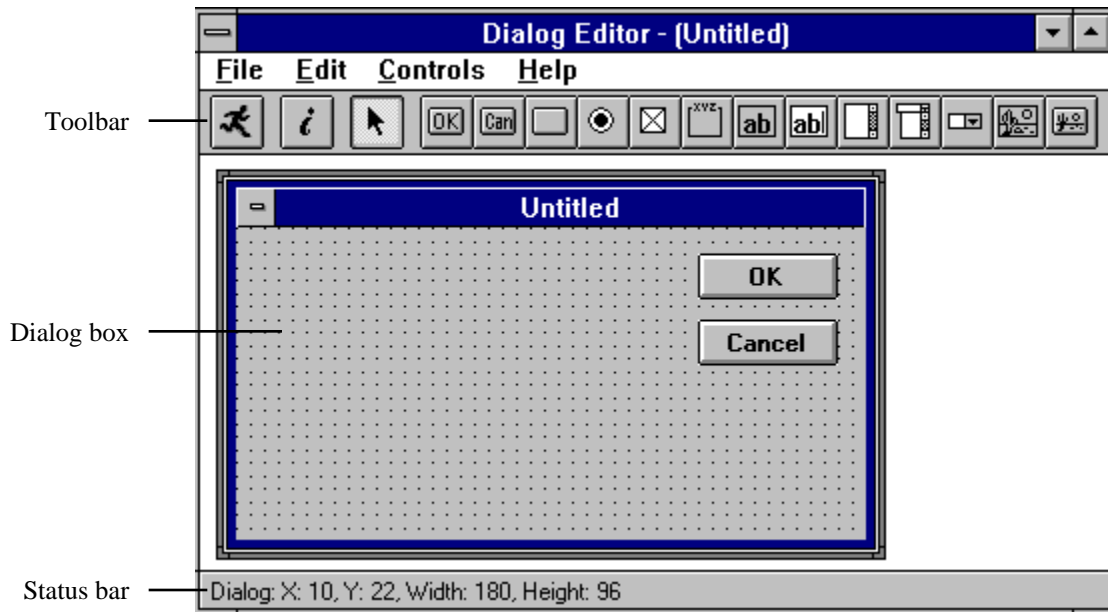
- Visual editing of a dialog box template in WM Basic
- The creation of dynamic dialog boxes

Using the Dialog Editor

This section presents general information that will help you work most effectively with Dialog Editor. It includes an overview of Dialog Editor's application window—the interface you'll use to create and edit dialog box templates in WM Basic—as well as a list of keyboard shortcuts and information on using the Help system.

Dialog Editor's Application Window

Before you begin creating a new custom dialog box, Dialog Editor's application window looks like this:



The application window contains the following elements:















- **Toolbar:** a collection of tools that you can use to provide instructions to Dialog Editor, as discussed in the following subsection
- **Dialog box:** the visual layout of the dialog box that you are currently creating or editing
- **Status bar:** provides key information about the operation you are currently performing, including the name of the currently selected control or dialog box, together with its position on the display and its dimensions; the name of a control you are about to add to the dialog box with the mouse pointer, together with the pointer's position on the display; the function of the currently selected menu command; and the activation of Dialog Editor's testing or capturing functions

Note: Dialog boxes created with Dialog Editor normally appear in an 8 point Helvetica font, both in Dialog Editor's application window and when the corresponding WM Basic code is run.

Dialog Editor's Toolbar

The following list briefly explains the purpose of each of the tools on Dialog Editor's toolbar, which you can use to add controls to your dialog box, make various changes to the dialog box and its controls, and test the dialog box's functioning.

Icon	Tool	Function
	Run	Runs the dialog box for testing purposes.
	Information	Displays the Information dialog box for the selected dialog box or control.

	Pick	Lets you select, move, and resize items and control the insertion point.
	OK Button	Adds an OK button to your dialog box.
	Cancel Button	Adds a Cancel button to your dialog box.
	Push Button	Adds a push button to your dialog box.
	Option Button	Adds an option button to your dialog box.
	Check Box	Adds a check box to your dialog box.
	Group Box	Adds a group box to your dialog box.
	Text	Adds a text control to your dialog box.
	Text Box	Adds a text box to your dialog box.
	List Box	Adds a list box to your dialog box.
	Combo Box	Adds a combo box to your dialog box.
	Drop List Box	Adds a drop list box to your dialog box.
	Picture	Adds a picture to your dialog box.
	Picture Button	Adds a picture button to your dialog box.

The types of dialog box controls that you can add with the control tools are fully described in the next section of the chapter.

Keyboard Shortcuts for Dialog Editor

The following keyboard shortcuts can be used for some of the operations you will perform most frequently in Dialog Editor.

Key(s)	Function
Alt+F4	Closes Dialog Editor's application window.
Ctrl+C	Copies the selected dialog box or control, without removing it from Dialog Editor's application window, and places it on the Clipboard.
Ctrl+D	Creates a duplicate copy of the selected control.

Ctrl+G	Displays the Grid dialog box.
Ctrl+I	Displays the Information dialog box for the selected dialog box or control.
Ctrl+V	Inserts the contents of the Clipboard into Dialog Editor. If the Clipboard contains WM Basic statements describing one or more controls, then Dialog Editor adds those controls to the current dialog box. If the Clipboard contains the WM Basic template for an entire dialog box, then Dialog Editor creates a new dialog box from the statements in the template.
Ctrl+X	Removes the selected dialog box or control from Dialog Editor's application window and places it on the Clipboard.
Ctrl+Z	Undoes the preceding operation.
Del	Removes the selected dialog box or control from Dialog Editor's application window without placing it on the Clipboard.
F1	Displays Help for the currently active window.
F2	Runs the dialog box for testing purposes.
F3	Sizes certain controls to fit the text they contain.
Shift+F1	Toggles the Help pointer.

Using the Help System

Dialog Editor provides several ways to obtain on-line help.

Here's how to display Help for the window or dialog box that is currently active.

To display Help for the currently active window:

- Press F1.
If Dialog Editor's application window was active, the Help system contents appear. If a dialog box was active, Help for that dialog box appears.

Here's how to access the Help system and search for a specific topic within it.

To pinpoint a specific topic in the Help system:

1. From the Help menu, choose the Search for Help on command.
A scrollable list of Help topics appears.
2. Select the desired topic from the list.
The topic you selected is displayed in a second scrollable list, together with closely related Help topics, if any.
3. If the desired topic is not already highlighted on the second list, select it and press Enter.

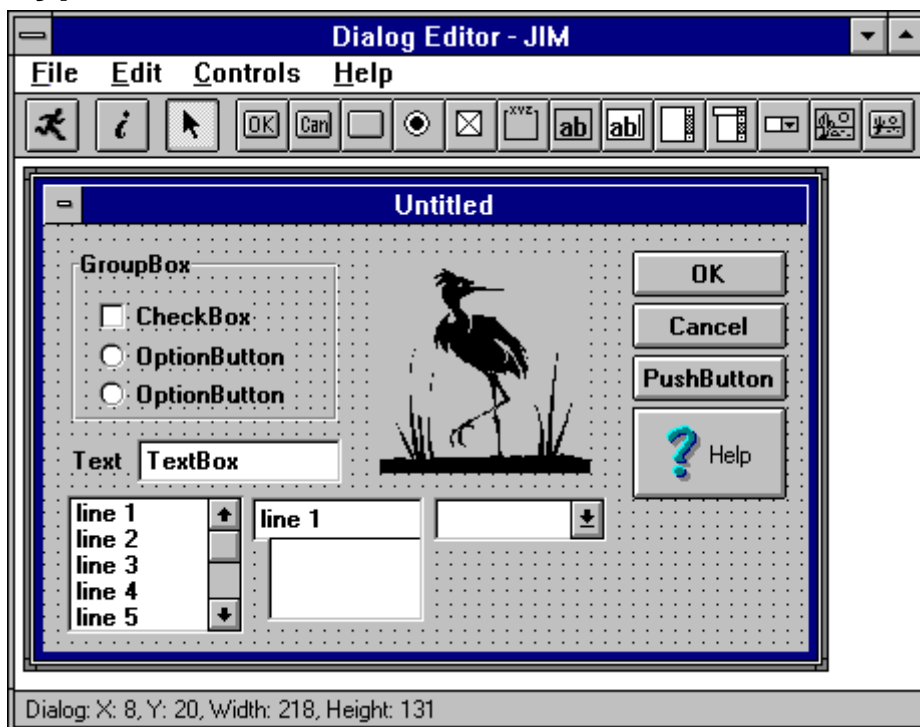
Help is displayed for the topic you selected.

Creating a Custom Dialog Box

This section describes the types of controls that Dialog Editor supports. It also explains how to create controls and initially position them within your dialog box, and offers some pointers on creating controls efficiently.

In the next section, "Editing a Custom Dialog Box," you'll learn how to make various types of changes to the controls that you've created—moving and resizing them, assigning labels and accelerator keys, and so forth.

Types of Controls



Dialog Editor supports the following types of standard Windows controls:

- **Push button:** a command button. The default OK and Cancel buttons are special types of push buttons.
- **Option button:** one of a group of two or more linked buttons that let users select only one from a group of mutually exclusive choices. A group of option buttons works the same way as the buttons on a car radio: because the buttons operate together as a group, clicking an unselected button in the group selects that button and automatically deselects the previously selected button in that group.
- **Check box:** a box that users can check or clear to indicate their preference regarding the alternative specified on the check box label.

- **Group box:** a rectangular design element used to enclose a group of related controls. You can use the optional group box label to display a title for the controls in the box.
- **Text:** a field containing text that you want to display for the users' information. The text in this field wraps, and the field can contain a maximum of 255 characters. Text controls can either display stand-alone text or be used as labels for text boxes, list boxes, combo boxes, drop list boxes, pictures, and picture buttons. You can choose the font in which the text appears.
- **Text box:** a field into which users can enter text (potentially, as much as 32K). By default, this field holds a single line of nonwrapping text. If you choose the Multiline setting in the Text Box Information dialog box, this field will hold multiple lines of wrapping text.
- **List box:** a displayed, scrollable list from which users can select one item. The currently selected item is highlighted on the list.
- **Combo box:** a text field with a displayed, scrollable list beneath it. Users can either select an item from the list or enter the name of the desired item in the text field. The currently selected item is displayed in the text field. If the item was selected from the scrolling list, it is highlighted there as well.
- **Drop list box:** a field that displays the currently selected item, followed by a downward-pointing arrow, which users can click to temporarily display a scrolling list of items. Once they select an item from the list, the list disappears and the newly selected item is displayed in the field.
- **Picture:** a field used to display a Windows bitmap or metafile.
- **Picture button:** a special type of push, or command, button on which a Windows bitmap or metafile appears.

Notes: Group boxes, text controls, and pictures are passive elements in a dialog box, inasmuch as they are used purely for decorative or informative purposes. Users cannot act upon these controls, and when they tab through the dialog box, the focus skips over these controls.

You can obtain a Windows bitmap or metafile from a file or from a specified library.

Adding Controls to a Dialog Box

In this subsection, you'll learn how to create controls and determine approximately where they first appear within your dialog box. In the following subsection, you'll learn how to determine the positioning of controls more precisely.

Here's how to add one or more controls to your dialog box using simple mouse and keyboard methods.

To add a control:

1. From the toolbar, choose the tool corresponding to the type of control you want to add.
When you pass the mouse pointer over an area of the display where a control can be placed, the pointer becomes an image of the selected

control with crosshairs (for positioning purposes) to its upper left. The name and position of the selected control appear on the status bar. When you pass the pointer over an area of the display where a control cannot be placed, the pointer changes into a circle with a slash through it (the "prohibited" symbol).

Note: You can only insert a control within the borders of the dialog box you are creating. You cannot insert a control on the dialog box's title bar or outside its borders.

2. Place the pointer where you want the control to be positioned and click the mouse button.

The control you just created appears at the specified location. (To be more specific, the upper left corner of the control will correspond to the position of the pointer's crosshairs at the moment you clicked the mouse button.) The control is surrounded by a thick frame, which means that it is selected, and it may also have a default label.

After the new control has appeared, the mouse pointer becomes an arrow, to indicate that the Pick tool is active and you can once again select any of the controls in your dialog box.

3. To add another control of the same type as the one you just added, press Ctrl+D.

A duplicate copy of the control appears.

To add a different type of control, repeat steps 1 and 2.

4. To reactivate the Pick tool, click the arrow-shaped tool on the toolbar.

-Or-

Place the mouse pointer on the title bar of the dialog box or outside the borders of the dialog box (that is, on any area where the mouse pointer turns into the "prohibited" symbol) and click the mouse button.

As you plan your dialog box, keep in mind that a single dialog box can contain no more than 255 controls and that a dialog box will not operate properly unless it contains either an OK button, a Cancel button, a push button, or a picture button. (When you create a new custom dialog box, an OK button and a Cancel button are provided for you by default.)

Later in the chapter, you'll learn more about selecting controls, and you'll learn how to assign labels.

Using the Grid to Help You Position Controls within a Dialog Box

The preceding subsection explained how to determine approximately where a newly created control will materialize in your dialog box. Here, you'll learn how to use Dialog Editor's grid to help you fine-tune the initial placement of controls.

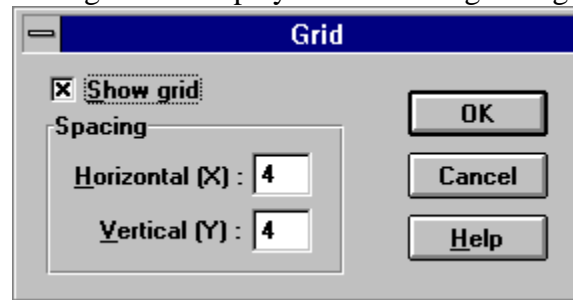
The area of your dialog box in which controls can be placed (that is, the portion of the dialog box below the title bar) can be thought of as a grid, with the X (horizontal) axis and the Y (vertical) axis intersecting in the upper left corner (the 0, 0 coordinates). The position of controls can be expressed in terms of X units with respect to the left border of this area and in terms of Y units with respect to the top border. (In fact, the position of controls is expressed in this manner within the dialog box template that you produce by working with Dialog Editor.)

Here's how to display the grid and adjust its X and Y settings, which can help you position controls more precisely within your dialog box.

To display and adjust the grid:

1. Press Ctrl+G.

Dialog Editor displays the following dialog box:



2. To display the grid in your dialog box, select the Show grid check box.
3. To change the current X and Y settings, enter new values in the X and Y fields.

Note: The values of X and Y in the Grid dialog box determine the grid's spacing. Assigning smaller X and Y values produces a more closely spaced grid, which enables you to move the mouse pointer in smaller horizontal and vertical increments as you position controls. Assigning larger X and Y values produces the opposite effect on both the grid's spacing and the movement of the mouse pointer. The X and Y settings entered in the Grid dialog box remain in effect regardless of whether you choose to display the grid.

4. Click the OK button or press Enter.

Dialog Editor displays the grid with the settings you specified. With the grid displayed, you can line up the crosshairs on the mouse pointer with the dots on the grid to position controls precisely and align them with respect to other controls.



As you move the mouse pointer over the dialog box after you have chosen a control tool from the toolbar, the status bar displays the name of the type of control you have selected and continually updates the position of the mouse pointer in X and Y units. (This information disappears if you move the mouse pointer over an area of the screen where a control cannot be placed.) After you click the mouse button to add a control, that control remains selected, and the status bar displays the control's width and height in dialog units as well as its name and position, as shown in the preceding illustration, in which the push button is selected.

Note: *Dialog units* represent increments of the font in which Dialog Editor creates dialog boxes (namely, 8 point Helvetica). Each X unit represents an increment equal to 1/4 of that font, and each Y unit represents an increment equal to 1/8 of that font.

Creating Controls Efficiently

Creating dialog box controls in random order might seem like the fastest approach. However, the order in which you create controls has some important implications, so a little advance planning can save you a lot of work in the long run.

Here are several points about creating controls that you should keep in mind:

- **Tabbing order:** Users can select dialog box controls by tabbing, as explained in the next subsection. The order in which you create the controls is what determines the tabbing order. That is, as users tab through the dialog box, the focus is changed from one control to the next in the order in which you created the controls (regardless of the order in which you position the controls in the dialog box). The closer you can come to creating controls in the order in which you want them to receive the tabbing focus, the fewer tabbing-order adjustments you'll have to make later on.
- **Option button grouping:** If you want a series of option buttons to work together as a mutually exclusive group, you must create all the buttons in that group one right after the other, in an unbroken sequence. If you get sidetracked and create a different type of control before you have finished creating all the option buttons in your group, you'll split the buttons into two (or more) separate groups. (Let's say you want to create an option button group with five buttons. You create three of the buttons and then create a list box, after which you finish creating the last two buttons. When you test your dialog box, you'll find that all five of these option buttons *don't* work together as a mutually exclusive group. Instead, the first three buttons will form one mutually exclusive group, and the last two buttons will form another mutually exclusive group.)
- **Accelerator keys:** As explained later in the chapter, you can provide easy access to a text box, list box, combo box, or drop list box by assigning an accelerator key to an associated text control, and you can provide easy access to the controls in a group box by assigning an accelerator key to the group box label. To do this, you must create the text control or group box first, followed immediately by the controls that you want to associate with it. If the controls are not created in the

correct order, they will not be associated in your dialog box template, and any accelerator key you assign to the text control or group box label will not work properly.

If you don't create controls in the most efficient order, the resulting problems with tabbing order, option button grouping, and accelerator keys usually won't become apparent until you test your dialog box. Although you can still fix these problems at that point, as explained later in the chapter, it will definitely be more cumbersome. In short, it's easier to prevent (or at least minimize) problems of this sort than to fix them after the fact.

Editing a Custom Dialog Box

In the preceding section, you learned how to create controls and determine where they initially appear within your dialog box. In this section, you'll learn how to make various types of changes to both the dialog box and the controls in it. The following topics are included:

- Selecting items so that you can work with them
- Using the Information dialog box to check and/or change various attributes of items
- Changing the position and size of items
- Changing titles and labels
- Assigning accelerator keys
- Specifying pictures
- Creating or modifying picture libraries under Windows
- Duplicating and deleting controls
- Undoing editing operations

Selecting Items

In order to edit a dialog box or a control, you must first select it. When you select an item, it becomes surrounded by a thick frame, as you saw in the preceding section.

Here's how to select a control using either mouse or keyboard methods.

To select a control:

- With the Pick tool active, place the mouse pointer on the desired control and click the mouse button.
- Or-
- With the Pick tool active, press the Tab key repeatedly until the focus moves to the desired control.
- The control is now surrounded by a thick frame to indicate that it is selected and you can edit it.

Here's how to select the entire dialog box using either mouse or keyboard methods.

To select the dialog box:

- With the Pick tool active, place the mouse pointer on the title bar of the dialog box or on an empty area within the borders of the dialog box (that is, on an area where there are no controls) and click the mouse button.
- Or-

With the Pick tool active, press the Tab key repeatedly until the focus moves to the dialog box.

The dialog box is now surrounded by a thick frame to indicate that it is selected and you can edit it.

Using the Information Dialog Box

The Information dialog box enables you to check and adjust various attributes of controls and dialog boxes. This subsection explains how to display the Information dialog box and provides an overview of the attributes with which it lets you work. In the following subsections, you'll learn more about how to use the Information dialog box to make changes to your dialog box and its controls.

Here's how to use the Dialog Box Information dialog box to check and adjust attributes that pertain to the dialog box as a whole.

To display the Information dialog box for a dialog box:

- With the Pick tool active, place the mouse pointer on an area of the dialog box where there are no controls and double-click the mouse button.

-Or-

With the Pick tool active, select the dialog box and either click the Information tool on the toolbar, press Enter, or press Ctrl+I.

Dialog Editor displays the Dialog Box Information dialog box:

The screenshot shows the 'Dialog Box Information' dialog box. It has a title bar with the text 'Dialog Box Information'. The dialog is divided into several sections. On the left, there is a 'Position' section with 'X:' and 'Y:' labels and input boxes. To its right is a 'Size' section with 'Width:' and 'Height:' labels and input boxes; the width is set to '164' and the height to '76'. Below these are three text input fields: 'Text:' with the value 'Untitled', 'Name:' with the value 'UserDialog', and 'Function:'. To the right of these fields are two checkboxes, both labeled 'Variable Name', each with an unchecked box. Below the 'Function' field is a 'Picture Library:' label and an input box, followed by a 'Browse...' button. On the right side of the dialog, there are three buttons: 'OK', 'Cancel', and 'Help'.

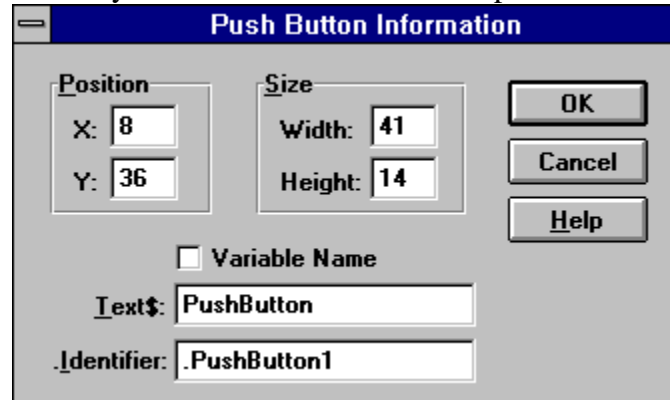
Here's how to use the Information dialog box for a control to check and adjust attributes that pertain to that particular control.

To display the Information dialog box for a control:

- With the Pick tool active, place the mouse pointer on the desired control and double-click the mouse button.

-Or-

With the Pick tool active, select the control and either click the Information tool on the toolbar, press Enter, or press Ctrl+I. Dialog Editor displays an Information dialog box corresponding to the control you selected. Here is an example:



The following lists show the attributes that you can change with the Dialog Box Information dialog box and the Information dialog boxes for the various controls. In some cases (specified below), it's mandatory to fill in the fields in which the attributes are specified—that is, you must either leave the default information in these fields or replace it with more meaningful information, but you can't leave the fields empty. In other cases, filling in these fields is optional—that is, you can either leave the default information in the fields, replace it with more meaningful information, or leave the fields entirely empty.

Note: A quick way to determine whether it's mandatory to fill in a particular Information dialog box field is to see whether the OK button becomes grayed out when you delete the information in that field. If it does, then you *must* fill in that field.

In many cases, you could simply leave the generic-sounding default information in the Information dialog box fields and worry about replacing it with more meaningful information after you paste the dialog box template into your script. However, if you take a few moments to replace the default information with something specific when you first create your dialog box, not only will you save yourself some work later on but you may also find that your changes make the WM Basic code produced by Dialog Editor more readily comprehensible and hence easier to work with.

Attributes That You Can Adjust with the Dialog Box Information Dialog Box

The Dialog Box Information dialog box can be used to check and adjust the following attributes, which pertain to the dialog box as a whole.

Mandatory/ Optional	Attribute
Optional	Position: X and Y coordinates on the display, in dialog units
Mandatory	Size: width and height of the dialog box, in dialog units

Optional	Style: options that allow you to determine whether the close box and title bar are displayed
Optional	Text\$: text displayed on the title bar of the dialog box
Mandatory	Name: name by which you refer to this dialog box template in your WM Basic code
Optional	.Function: name of a WM Basic function in your dialog box
Optional	Picture Library: picture library from which one or more pictures in the dialog box are obtained

Attributes That You Can Adjust with the Information Dialog Box for a Control

The Information dialog box for a control can be used to check and adjust the following attributes. The second column of the list indicates the control(s) to which each attribute pertains.

Mandatory/ Optional	Control(s) Affected	Attribute
Mandatory	All controls	Position: X and Y coordinates within the dialog box, in dialog units
Mandatory	All controls	Size: width and height of the control, in dialog units
Optional	Push button, option button, check box, group box, and text	Text\$: text displayed on a control
Optional	Text	Font: font in which text is displayed
Optional	Text box	Multiline: option that allows you to determine whether users can enter a single line of text or multiple lines
Optional	OK button, Cancel button, push button, option button, group box, text, picture, and picture button	.Identifier: name by which you refer to a control in your WM Basic code

Mandatory	Check box, text box, list box, combo box, and drop list box	.Identifier: name by which you refer to a control in your WM Basic code; also contains the result of the control after the dialog box has been processed
Optional	Picture, picture button	.Identifier: name of the file containing a picture that you want to display or the name of a picture that you want to display from a specified picture library
Optional	Picture	Frame: option that allows you to display a 3-D frame
Mandatory	List box, combo box, and drop list box	Array\$: name of an array variable in your WM Basic code
Mandatory	Option button	.Option Group: name by which you refer to a group of option buttons in your WM Basic code

Changing Position and Size

This subsection explains how Dialog Editor helps you keep track of the location and dimensions of dialog boxes and controls, and presents several ways to move and resize these items.

Keeping Track of Position and Size

Dialog Editor's display can be thought of as a grid, in which the X (horizontal) axis and the Y (vertical) axis intersect in the upper left corner of the display (the 0, 0 coordinates). The position of the dialog box you are creating can be expressed in terms of X units with respect to the left border of the parent window and in terms of Y units with respect to the top border. As explained in the preceding section, the portion of your dialog box below the title bar can also be thought of as a grid, with the X and Y axes intersecting in the upper left corner of this area. The position of controls within the dialog box can be expressed in terms of X units with respect to the left border of this area and in terms of Y units with respect to the top border. When you select a dialog box or control, the status bar displays its position in X and Y units as well as its width and height in dialog units. Each time you move or resize an item, the corresponding information on the status bar is updated. You can use this information to position and size items more precisely.

Changing the Position of an Item

Dialog Editor provides several ways to reposition dialog boxes and controls. Here's how to move a dialog box or control by dragging it with the mouse.

To reposition an item with the mouse:

1. With the Pick tool active, place the mouse pointer on an empty area of the dialog box or on a control.
2. Depress the mouse button and drag the dialog box or control to the desired location.

Note: The increments by which you can move a control with the mouse are governed by the grid setting. For example, if the grid's X setting is 4 and its Y setting is 6, you'll be able to move the control horizontally only in increments of 4 X units and vertically only in increments of 6 Y units. This feature is handy if you're trying to align controls in your dialog box. If you want to move controls in smaller or larger increments, press Ctrl+G to display the Grid dialog box and adjust the X and Y settings.

Here's how to move a selected dialog box or control by pressing the arrow keys.

To reposition an item with the arrow keys:

1. Select the dialog box or control that you want to move.
2. Press an arrow key once to move the item by 1 X or Y unit in the desired direction.

-Or-

Depress an arrow key to "nudge" the item steadily along in the desired direction.

Note: When you reposition an item with the arrow keys, a faint, partial afterimage of the item may remain visible in the item's original position. These afterimages are rare and will disappear once you test your dialog box.

Here's how to move a selected dialog box by changing its coordinates in the Dialog Box Information dialog box.

To reposition a dialog box with the Dialog Box Information dialog box:

1. Display the Dialog Box Information dialog box.
2. Change the X and Y coordinates in the Position group box.

-Or-

Leave the X and/or Y coordinates blank.

3. Click the OK button or press Enter.

If you specified X and Y coordinates, the dialog box moves to that position. If you left the X coordinate blank, the dialog box will be centered horizontally relative to the parent window of the dialog box when the dialog box is run. If you left the Y coordinate blank, the dialog box will be centered vertically relative to the parent window of the dialog box when the dialog box is run.

Here's how to move a selected control by changing its coordinates in the Information dialog box for that control.

To reposition a control with the Information dialog box:

1. Display the Information dialog box for the control that you want to move.
2. Change the X and Y coordinates in the Position group box.
3. Click the OK button or press Enter.

The control moves to the specified position.

Notes: When you move a dialog box or control with the arrow keys or with the Information dialog box, the item's movement is *not* restricted to the increments specified in the grid setting.

When you attempt to test a dialog box containing hidden controls (i.e., controls positioned entirely outside the current borders of your dialog box), Dialog Editor displays a message advising you that there are controls outside the dialog box's borders and asks whether you wish to proceed with the test. If you proceed, the hidden controls will be disabled for testing purposes. (Testing dialog boxes is discussed later in the chapter.)

Changing the Size of an Item

Dialog boxes and controls can be resized either by directly manipulating them with the mouse or by using the Information dialog box. Certain controls can also be resized automatically to fit the text displayed on them.

Here's how to change the size of a selected dialog box or control by dragging its borders or corners with the mouse.

To resize an item with the mouse:

1. With the Pick tool active, select the dialog box or control that you want to resize.
2. Place the mouse pointer over a border or corner of the item.
3. Depress the mouse button and drag the border or corner until the item reaches the desired size.

Here's how to change the size of a selected dialog box or control by changing its Width and/or Height settings in the Information dialog box.

To resize an item with the Information dialog box:

1. Display the Information dialog box for the dialog box or control that you want to resize.
2. Change the Width and Height settings in the Size group box.
3. Click the OK button or press Enter.

The dialog box or control is resized to the dimensions you specified.

Here's how to adjust the borders of certain controls automatically to fit the text displayed on them.

To resize selected controls automatically:

1. With the Pick tool active, select the option button, text control, push button, check box, or text box that you want to resize.
2. Press F2.

The borders of the control will expand or contract to fit the text displayed on it.

Note: Windows metafiles always expand or contract proportionally to fit within the picture control or picture button control containing them. In contrast, windows bitmaps are of a fixed size. If you place a bitmap in a control that is smaller than the bitmap, the bitmap is clipped off on the right and bottom. If you place a bitmap in a control that is larger than the bitmap, the bitmap is centered within the borders of the control. Picture controls and picture button controls must be resized manually.

Changing Titles and Labels

By default, when you begin creating a dialog box, its title reads "Untitled," and when you first create group boxes, option buttons, push buttons, text controls, and check boxes, they have generic-sounding default labels, such as "Group Box" and "Option Button." Here's how to change the title of your dialog box as well as the labels of group boxes, option buttons, push buttons, text controls, and check boxes.

To change a dialog box title or a control label:

1. Display the Information dialog box for the dialog box whose title you want to change or for the control whose label you want to change.
2. Enter the new title or label in the Text\$ field.

Note: Dialog box titles and control labels are optional. Therefore, you can leave the Text\$ field blank.

3. If the information in the Text\$ field should be interpreted as a variable name rather than a literal string, select the Variable Name check box.
4. Click the OK button or press Enter.
The new title or label is now displayed on the title bar or on the control.

Although OK and Cancel buttons also have labels, you cannot change them. The remaining controls (text boxes, list boxes, combo boxes, drop list boxes, pictures, and picture buttons) don't have their own labels, but you can position a text control above or beside these controls to serve as a de facto label for them.

Assigning Accelerator Keys

Accelerator keys enable users to access dialog box controls simply by pressing Alt + a specified letter. Users can employ accelerator keys to choose a push button or an option button; toggle a check box on or off; and move the insertion point into a text box or group box or to the currently selected item in a list box, combo box, or drop list box.

An accelerator key is essentially a single letter that you designate for this purpose from a control's label. You can assign an accelerator key directly to controls that have their own label (option buttons, push buttons, check boxes, and group boxes). (You can't assign an accelerator key to OK and Cancel buttons because, as noted above, their labels can't be edited.) You can create a de facto accelerator key for certain controls that don't have their own labels (text boxes, list boxes, combo boxes, and drop list boxes) by assigning an accelerator key to an associated text control.

Here's how to designate a letter from a control's label to serve as the accelerator key for that control.

To assign an accelerator key:

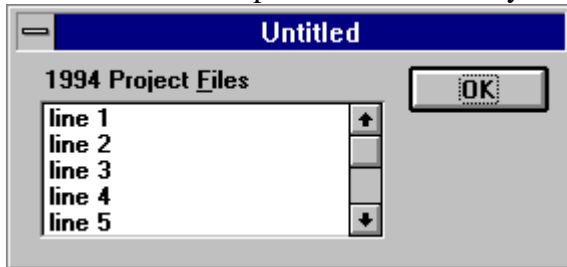
1. Display the Information dialog box for the control to which you want to assign an accelerator key.
2. In the Text\$ field, type an ampersand (&) before the letter you want to designate as the accelerator key.
3. Click the OK button or press Enter.
The letter you designated is now underlined on the control's label, and users will be able to access the control by pressing Alt + the underlined letter.

Note: Accelerator key assignments must be unique within a particular dialog box. If you attempt to assign the same accelerator key to more than one control, Dialog Editor displays a reminder that that letter has already been assigned.

If, for example, you have a push button whose label reads *Apply*, you can designate *A* as the accelerator key by displaying the Push Button Information dialog box and typing *&Apply* in the Text\$ field. When you press Enter, the button label looks like the following illustration, and users will be able to choose the button by typing Alt+A.



As another example, let's say you have a list box that is immediately preceded in the dialog box template by a text control whose label reads *1994 Project Files*. By using the method described above, you can designate *F* as the accelerator key. When you click OK or press Enter, the text control label looks like the following illustration, and users will be able to move the insertion point to the currently selected item in the list box by typing Alt+F.



Note: In order for such a de facto accelerator key to work properly, the text control or group box label to which you assign the accelerator key *must* be associated with the control(s) to which you want to provide user access—that is, in the dialog box template, the description of the text control or group box must immediately precede the description of the control(s) that you want associated with it. The simplest way to establish such an association is to create the text control or group box first, followed immediately by the associated control(s).

Specifying Pictures

In the preceding section, you learned how to add picture controls and picture button controls to your dialog box. But these controls are nothing more than empty outlines until you specify the pictures that you want them to display.

A picture control or picture button control can display a Windows bitmap or metafile, which you can obtain from a file or from a specified library. (Refer to the following subsection for information on creating or modifying picture libraries under Windows.)

Here's how to display a Windows bitmap or metafile from a file on a picture control or picture button control by using the control's Information dialog box to indicate the file in which the picture is contained.

To specify a picture from a file:

1. Display the Information dialog box for the picture control or picture button control whose picture you want to specify.
2. In the Picture source option button group, select File.
3. In the Name\$ field, enter the name of the file containing the picture you want to display in the picture control or picture button control.

Note: By clicking the Browse button, you can display the Select a Picture File dialog box and use it to find the file.

4. Click the OK button or press Enter.
The picture control or picture button control now displays the picture you specified.

Here's how to display a Windows bitmap or metafile from a library on a picture control or picture button control by first using the Dialog Box Information dialog box to specify the library and then using the control's Information dialog box to indicate the name of the picture.

To specify a picture from a picture library:

1. Display the Dialog Box Information dialog box.
2. In the Picture Library field, specify the name of the picture library that contains the picture(s) you want to display in your dialog box.

Notes: By clicking the Browse button, you can display the Select a Picture Library dialog box and use it to find the library.

If you specify a picture library in the Dialog Box Information dialog box, all the pictures in your dialog box must come from this library.

3. Click the OK button or press Enter.
4. Display the Information dialog box for the picture control or picture button control whose picture you want to specify.
5. In the Picture source option button group, select Library.
6. In the Name\$ field, enter the name of the picture you want to display on the picture control or picture button control. (This picture must be from the library that you specified in step 2.)
7. Click the OK button or press Enter.
The picture control or picture button control now displays the picture you specified.

Creating or Modifying Picture Libraries under Windows

The `Picture` statement in WM Basic allows images to be specified as individual picture files or as members of a picture library, which is a DLL that contains a collection of pictures.

Currently, both Windows bitmaps and metafiles are supported. You can obtain a picture library either by creating a new one or by modifying an existing one, as described below. Each image is placed into the DLL as a resource identified by its unique resource identifier. This identifier is the name used in the `Picture` statement to specify the image. The following resource types are supported in picture libraries:

Resource Type	Description
2	Bitmap. This is defined in <code>windows.h</code> as <code>RT_BITMAP</code> .
256	Metafile. Since there is no resource type for metafiles, 256 is used.

Here's how to create a new picture library to contain the Windows bitmaps or metafiles that you want to display on picture controls or picture button controls in your dialog box.

To create a picture library under Windows:

1. Create a C file containing the minimal code required to establish a DLL. The following code can be used:

```
#include <windows.h>
int CALLBACK LibMain(
    HINSTANCE hInstance,
    WORD wDataSeg,
    WORD wHeapSz,
    LPSTR lpCmdLine) {
    UnlockData(0);
    return 1;
}
```
2. Use the following code to create a DEF file for your picture library:

```
LIBRARY
DESCRIPTION "My Picture Library"
EXETYPE WINDOWS
CODE LOADONCALL MOVABLE DISCARDABLE
DATA PRELOAD MOVABLE SINGLE
HEAPSIZE 1024
```
3. Create a resource file containing your images. The following example shows a resource file using a bitmap called `sample.bmp` and a metafile called `usa.wmf`.

```
#define METAFILE 256
USA METAFILE "usa.wmf"
MySample BITMAP "sample.bmp"
```
4. Create a make file that compiles your C module, creates the resource file, and links everything together.

Here's how to modify an existing picture library to contain the Windows bitmaps or metafiles that you want to display on picture controls or picture button controls in your dialog box.

To modify an existing picture library:

1. Make a copy of the picture library you want to modify.
2. Modify the copy by adding images using a resource editor such as Borland's Resource Workshop or Microsoft's App Studio.

Note: When you use a resource editor, you need to create a new resource type for metafiles (with the value 256).

Duplicating and Deleting Controls

Here's how to use Dialog Editor's duplicating feature, which saves you the work of creating additional controls individually if you need one or more copies of a particular control.

To duplicate a control:

1. Select the control that you want to duplicate.
2. Press Ctrl+D.
A duplicate copy of the selected control appears in your dialog box.
3. Repeat step 2 as many times as necessary to create the desired number of duplicate controls.

Duplicating is a particularly efficient approach if you need to create a group of controls, such as a series of option buttons or check boxes. Simply create the first control in the group and then, while the newly created control remains selected, repeatedly press Ctrl+D until you have created the necessary number of copies.

Dialog Editor also enables you to delete single controls or even clear the entire dialog box. If you want to remove controls from your dialog box selectively, here's how to delete them one at a time.

To delete a single control:

1. Select the control you want to delete.
2. Press Del.

The selected control is removed from your dialog box.

If you want to "wipe the slate clean" and start all over again with your dialog box, here's how to remove all its controls in a single operation.

To delete all the controls in a dialog box:

1. Select the dialog box.
2. Press Del.
3. If the dialog box contains more than one control, Dialog Editor prompts you to confirm that you want to delete all controls. Click the Yes button or press Enter.
All the controls disappear, but the dialog box's title bar and close box (if displayed) remain unchanged.

Undoing Editing Operations

You can undo editing operations that produce a change in your dialog box, including:

- The addition of a control
- The insertion of one or more controls from the Clipboard
- The deletion of a control
- Changes made to a control or dialog box, either with the mouse or with the Information dialog box

You cannot undo operations that don't produce any change in your dialog box, such as selecting controls or dialog boxes and copying material to the Clipboard.

Here's how to reverse the effect of the preceding editing operation.

To undo an editing operation:

- Press Ctrl+Z.

Your dialog box is restored to the way it was before you performed the editing operation.

Editing an Existing Dialog Box

There are three ways to edit an existing dialog box: (1) You can copy the WM Basic template of the dialog box you want to edit from a script to the Clipboard and paste it into Dialog Editor. (2) You can use the capture feature to "grab" an existing dialog box from another application and insert a copy of it into Dialog Editor. (3) You can open a dialog box template file that has been saved on a disk. Once you have the dialog box displayed in Dialog Editor's application window, you can edit it using the methods described earlier in the chapter.

Pasting an Existing Dialog Box into Dialog Editor

You can use Dialog Editor to modify the WM Basic statements in your script that correspond to an entire dialog box or to one or more dialog box controls.

If you want to modify a WM Basic dialog box template contained in your script, here's how to select the template and paste it into Dialog Editor for editing.

To edit an existing dialog box directly:

1. Select the entire WM Basic dialog box template (from the `Begin Dialog` instruction to the `End Dialog` instruction) in your script.
2. Choose **Edit Dialog** in the **Edit** menu of the Script Editor.
Dialog Editor creates a new dialog box corresponding to the template selected in the Script Editor.

To paste an existing dialog box into Dialog Editor:

1. Copy the entire WM Basic dialog box template (from the `Begin Dialog` instruction to the `End Dialog` instruction) from your script to the Clipboard.
2. Open Dialog Editor.
3. Press **Ctrl+V**.
4. When Dialog Editor asks whether you want to replace the existing dialog box, click the **Yes** button.
Dialog Editor creates a new dialog box corresponding to the template contained on the Clipboard.

If you want to modify the WM Basic statements in your script that correspond to one or more dialog box controls, here's how to select the statements and paste them into Dialog Editor for editing.

To paste one or more controls from an existing dialog box into Dialog Editor:

1. Copy the WM Basic description of the control(s) from your script to the Clipboard.
2. Open Dialog Editor.
3. Press **Ctrl+V**.
Dialog Editor adds to your current dialog box one or more controls corresponding to the description contained on the Clipboard.

Notes: When you paste a dialog box template into Dialog Editor, the tabbing order of the controls is determined by the order in which the controls are described in the template. When you paste one or more controls into Dialog Editor, they will come last in the tabbing order, following the controls that are already present in the current dialog box.


If there are any errors in the WM Basic statements that describe the dialog box or controls, the Dialog Translation Errors dialog box will appear when you attempt to paste these statements into Dialog Editor. This dialog box shows the lines of code containing the errors and provides a brief description of the nature of each error.

Capturing a Dialog Box from Another Application


Here's how to capture the standard Windows controls from any standard Windows dialog box in another application and insert those controls into Dialog Editor for editing.

To capture an existing standard Windows dialog box:

1. Display the dialog box you want to capture.
2. Open Dialog Editor.
3. Choose the Capture Dialog command from the File menu.
Dialog Editor's application window moves behind all other open application windows, and the dialog box you displayed in step 1 reappears. The mouse pointer, previously an arrow, now looks like a butterfly net.
4. Place the mouse pointer over the dialog box that you want to capture.
If the mouse pointer is over a standard Windows dialog box that contains some standard Windows controls, a tiny dialog box appears in front of the mouse pointer's butterfly net to indicate that the pointer has found controls that can be captured. If the mouse pointer is not over a standard Windows dialog box that contains standard Windows controls, the butterfly net remains unchanged to indicate that the mouse pointer has not found controls that can be captured.



Mouse pointer positioned over an area of the screen that does not contain standard Windows controls



Mouse pointer positioned over a standard Windows dialog box that contains some standard Windows controls
5. Click the mouse button.
Dialog Editor's application window moves in front of all other open application windows and now displays the standard Windows controls from the target dialog box.

Note: Dialog Editor only supports standard Windows controls and standard Windows dialog boxes. Therefore, if the target dialog box contains both standard Windows controls and custom controls, only the standard Windows controls will appear in Dialog Editor's application window. If the target dialog box is not a standard Windows dialog box, you will be unable to capture the dialog box or any of its controls.

Opening a Dialog Box Template File

Here's how to open any dialog box template file that has been saved on a disk so you can edit the template in Dialog Editor.

To open a dialog box template file:

1. Choose Open from the File menu.
The Open Dialog File dialog box appears.
2. Select the file containing the dialog box template that you want to edit and click the OK button.
Dialog Editor creates a dialog box from the statements in the template and displays it in the application window.

Note: If there are any errors in the WM Basic statements that describe the dialog box, the Dialog Translation Errors dialog box will appear when you attempt to load the file into Dialog Editor. This dialog box shows the lines of code containing the errors and provides a brief description of the nature of each error.

Testing an Edited Dialog Box

Dialog Editor lets you run your edited dialog box for testing purposes. When you click the Test tool, your dialog box comes alive, which gives you an opportunity to make sure it functions properly and fix any problems before you incorporate the dialog box template into your script.

Before you run your dialog box, take a moment to look it over for basic problems such as the following:

- Does the dialog box contain a command button—that is, a default OK or Cancel button, a push button, or a picture button?
- Does the dialog box contain all the necessary push buttons?
- Does the dialog box contain a Help button if one is needed?
- Are the controls aligned and sized properly?
- If there is a text control, is its font set properly?
- Are the close box and title bar displayed (or hidden) as you intended?
- Are the control labels and dialog box title spelled and capitalized correctly?
- Do all the controls fit within the borders of the dialog box?
- Could you improve the design of the dialog box by adding one or more group boxes to set off groups of related controls?
- Could you clarify the purpose of any unlabeled control (such as a text box, list box, combo box, drop list box, picture, or picture button) by adding a text control to serve as a de facto label for it?
- Have you made all the necessary accelerator key assignments?

After you've fixed any elementary problems, you're ready to run your dialog box so you can check for problems that don't become apparent until a dialog box is activated.

Testing your dialog box is an iterative process that involves running the dialog box to see how well it works, identifying problems, stopping the test and fixing those problems, then running the dialog box again to make sure the problems are fixed and to identify any

additional problems, and so forth—until the dialog box functions the way you intend. Here's how to test your dialog box and fine-tune its performance.

To test your dialog box:

1. Click the Run tool on the toolbar.
-Or-
Press F5.
The dialog box becomes operational, and you can check how it functions.
2. To stop the test, click the Run tool, press F5, or double-click the dialog box's close box (if it has one).
3. Make any necessary adjustments to the dialog box.
4. Repeat steps 1–3 as many times as you need in order to get the dialog box working properly.

When testing a dialog box, you can check for operational problems such as the following:

- **Tabbing order:** When you press the Tab key, does the focus move through the controls in a logical order? (Remember, the focus skips over items that users cannot act upon, including group boxes, text controls, and pictures.)
When you paste controls into your dialog box, Dialog Editor places their descriptions at the end of your dialog box template, in the order in which you paste them in. Therefore, you can use a simple cut-and-paste technique to adjust the tabbing order. First, click the Run tool to end the test and then, proceeding in the order in which you want the controls to receive the focus, select each control, cut it from the dialog box (by pressing Ctrl+X), and immediately paste it back in again (by pressing Ctrl+V). The controls will now appear in the desired order in your template and will receive the tabbing focus in that order.
- **Option button grouping:** Are the option buttons grouped correctly? Does selecting an unselected button in a group automatically deselect the previously selected button in that group?
To merge two groups of option buttons into a single group, click the Run tool to end the test and then use the Option Button Information dialog box to assign the same .Option Group name for all the buttons that you want included in that group.
- **Text box functioning:** Can you enter only a single line of nonwrapping text, or can you enter multiple lines of wrapping text? If the text box doesn't behave the way you intended, click the Run tool to end the test; then display the Text Box Information dialog box and select or clear the Multiline check box.
- **Accelerator keys:** If you have assigned an accelerator key to a text control or group box in order to provide user access to a text box, list box, combo box, drop list box, or group box, do the accelerator keys work properly? That is, if you press Alt + the designated accelerator key, does the insertion point move into the text box or group box or to the currently selected item in the list box, combo box, or drop list box?

If the accelerator key doesn't work properly, it means that the text box, list box, combo box, drop list box, or group box is not associated with the text control or group box to which you assigned the accelerator key—that is, in your dialog box template, the description of the text control or group box does not immediately precede the description of the control(s) that should be associated with it. As with tabbing-order problems (discussed above), you can fix this problem by using a simple cut-and-paste technique to adjust the order of the control descriptions in your template. First, click the Run tool to end the test; then cut the text control or group box from the dialog box and immediately paste it back in again; and finally, do the same with each of the controls that should be associated with the text control or group box. The controls will now appear in the desired order in your template, and the accelerator keys will work properly.

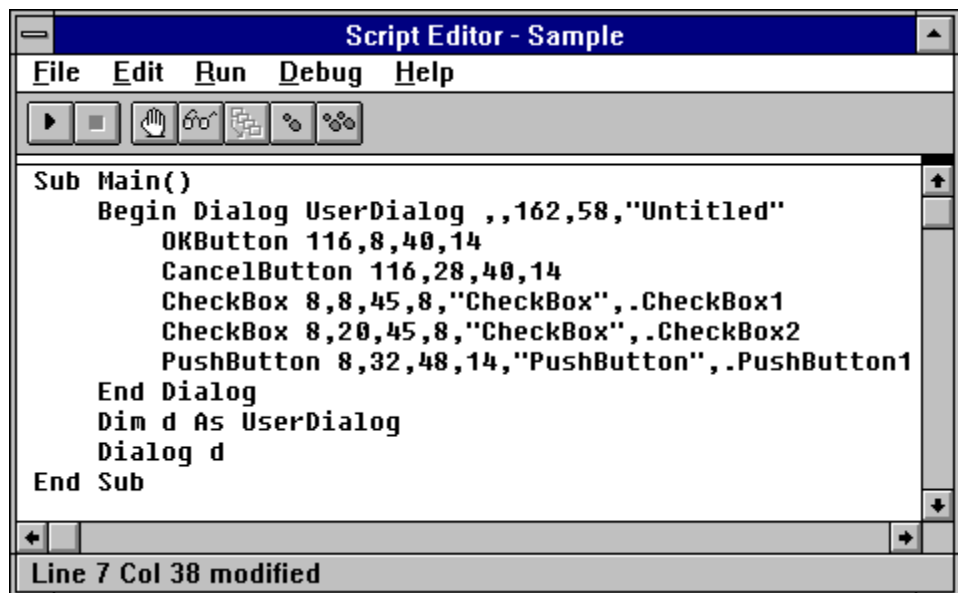
Incorporating a Dialog Box Template into Your Script

Dialog boxes and dialog box controls are communicated between Dialog Editor and your script via the Clipboard, where they are represented as WM Basic statements. Here's how to copy a dialog box or control and paste it into your script.

To incorporate a dialog box or control into your script:

1. Select the dialog box or control that you want to incorporate into your script.
2. Press Ctrl+C.
3. Open your script and paste in the contents of the Clipboard at the desired point.

The dialog box template or control is now described in WM Basic statements in your script, as shown in the following example.



Exiting from Dialog Editor

Here's how to get out of Dialog Editor. When you exit, you can save your dialog box template (that is, the WM Basic description of the dialog box) as a text file.


To exit from Dialog Editor:

1. Press Alt+F4.
If you have made changes to your dialog box template, Dialog Editor asks whether you want to save those changes.
2. If you want to save your changes to a text file, click the Yes button.
Dialog Editor displays the Save Dialog File dialog box, which you can use to specify the file to which you want to save your template.

Menu/Tools Reference


File Menu

Menu Command	Toolbar Tool	Keyboard Shortcut	Function
<u>N</u> ew			Creates a new dialog box. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.





<u>O</u> pen...			Displays the Open Dialog File dialog box, which you can use to open an existing dialog box template. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.
<u>S</u> ave			<p>If you have already created a file for the current dialog box template, saves the template to that file.</p> <p>If you have not yet created a file for the current dialog box template, displays the Save Dialog File dialog box, which you can use to specify the file to which you want to save the current template.</p>
<u>S</u> ave <u>A</u> s...			Displays the Save Dialog File dialog box, which you can use to save the current dialog box template in a file under a new name.
<u>T</u> est Dialog		F5	Toggles between the run mode (in which the dialog box "comes alive" for testing purposes) and the edit mode (in which you can make changes to the dialog box).
<u>C</u> apture Dialog			Captures the standard Windows controls from a standard Windows dialog box in another Windows application.
<u>E</u> xit		Alt+F4	Closes Dialog Editor. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.
Exit & Return		Alt+F4	Closes Dialog Editor and returns you to the host application. Dialog Editor prompts you before discarding any changes you have made to your current dialog box.







Edit Menu




Menu Command	Toolbar Tool	Keyboard Shortcut	Function
<u>U</u> ndo		Ctrl+Z	Allows you to undo up to 10 preceding operations. The Undo command continually indicates the next operation you can undo by selecting it and grays out when there are no more operations that can be undone.
Cu <u>t</u>		Ctrl+X	Removes the selected dialog box or control from Dialog Editor's application window and places it on the Clipboard.
<u>C</u> opy		Ctrl+C	Copies the selected dialog box or control, without removing it from Dialog Editor's application window, and places it on the Clipboard.
<u>P</u> aste		Ctrl+V	<p>Inserts the contents of the Clipboard into Dialog Editor.</p> <p>If the Clipboard contains WM Basic statements describing one or more controls, then those controls are added to the current dialog box. If the Clipboard contains the WM Basic template for an entire dialog box, then Dialog Editor creates a new dialog box from the statements in the template.</p> <p>If the WM Basic statements contain errors, Dialog Editor displays the Dialog Translation Errors dialog box, which helps you pinpoint the location and nature of the errors.</p>

<u>D</u> el	Del	Removes the selected dialog box or control from Dialog Editor's application window without placing it on the Clipboard. If you have selected the dialog box and it contains more than one control, Dialog Editor prompts you before removing all the controls from it.
Duplicate	Ctrl+D	Creates a duplicate copy of the selected control.
Size to Text	F3	Adjusts the borders of certain controls to fit the text displayed on them.
<u>I</u> no...	 Ctrl+I	Displays the Information dialog box for the selected dialog box or control. You can use this dialog box to check and adjust various attributes of controls and dialog boxes.
<u>G</u> rid...	Ctrl+G	Displays the Grid dialog box, which you can use to display or turn off the grid and adjust the grid's spacing.

Controls Menu

Menu Command	Toolbar Tool	Function
<u>O</u> K button		Adds a default OK button to your dialog box. An OK button is a special type of push, or command, button.
<u>C</u> ancel button		Adds a default Cancel button to your dialog box. A Cancel button is a special type of push, or command, button.
<u>P</u> ush button		Adds a push, or command, button to your dialog box.
<u>O</u> ption button		Adds an option button to your dialog box. An option button is one of a group of two or more linked buttons that let users select only one from a group of mutually exclusive choices.


<u>C</u> heck box		Adds a check box to your dialog box. Users can check or clear a check box to indicate their preference regarding the alternative specified on the check box label.
<u>G</u> roup box		Adds a group box to your dialog box. A group box is a rectangular design element used to enclose a group of related controls. You can use the optional group box label to display a title for the controls in the box.
<u>T</u> ext		Adds a text control to your dialog box. A text control is a field containing text that you want to display for the users' information. The text in this field wraps, and the field can contain a maximum of 255 characters. Text controls can either display stand-alone text or be used as labels for text boxes, list boxes, combo boxes, drop list boxes, pictures, and picture buttons. You can choose the font in which the text appears.
<u>T</u> ext box		Adds a text box to your dialog box. A text box is a field into which users can enter text (potentially, as much as 32K). By default, this field holds a single line of nonwrapping text. If you choose the Multiline setting in the Text Box Information dialog box, this field will hold multiple lines of wrapping text.
<u>L</u> ist box		Adds a list box to your dialog box. A list box is a displayed, scrollable list from which users can select one item. The currently selected item is highlighted on the list.
<u>C</u> om <u>b</u> o box		Adds a combo box to your dialog box. A combo box consists of a text field with a displayed, scrollable list beneath it. Users can either select an item from the list or enter the name of the desired item in the text field. The currently selected item is displayed in the text field. If the item was selected from the scrolling list, it is highlighted there as well.

<u>D</u> rop list box		Adds a drop list box to your dialog box. A drop list box consists of a field that displays the currently selected item, followed by a downward-pointing arrow, which users can click to temporarily display a scrolling list of items. Once they select an item from the list, the list disappears and the newly selected item is displayed in the field.
P <u>i</u> cture		Adds a picture to your dialog box. A picture is a field used to display a Windows bitmap or metafile.
P <u>i</u> cture button		Adds a picture button to your dialog box. A picture button is a special type of push, or command, button on which a Windows bitmap or metafile appears.

Help Menu

Menu Command	Keyboard Shortcut	Function
<u>C</u> ontents	F1	Presents a list of major topics in the Help system. By clicking a topic on the list, you can display the available Help information about that topic.
<u>S</u> earch for Help On...		Displays a dialog box that allows users to search for Help topics containing specific keywords.
<u>A</u> bout Dialog Editor...		Displays the About Dialog Editor dialog box, which indicates application name, version number, copyright statement, and icon, as well as additional information such as amount of available memory and disk space.

Pick Tool

Toolbar Tool	Function
Pick 	Lets you select, move, and resize items and control the insertion point.

Using a Custom Dialog

You can use a custom dialog box to display information to a user while providing an opportunity to respond. After using Dialog Editor to insert a dialog box template into your script, you'll need to make the following modifications to your script.

1. Create a *dialog record* by using a Dim statement.
2. Put information into the custom dialog box by assigning values to dialog box controls.
3. Display the custom dialog box by using either the Dialog() function or the Dialog statement.
4. Retrieve values from the custom dialog box after the user closes it.

Creating a Dialog Record

To store the values retrieved from a customer dialog box, you create a dialog record with a Dim statement, using the following syntax:

```
Dim DialogRecord As DialogVariable
```

Here are some examples of how to create dialog records:

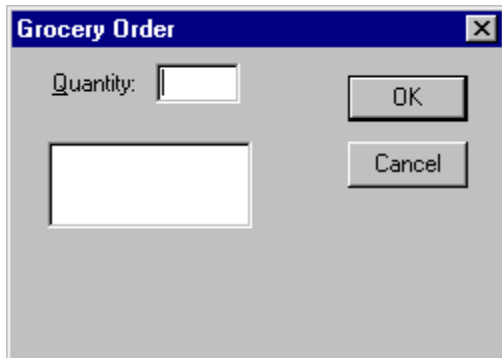
```
Dim b As UserDialog          ' Define a dialog record "b"
Dim PlayCD As CDDialog      ' Define a dialog record "PlayCD"
```

Here is a sample script named DIALOG1.WBS that illustrates how to create a dialog record named `b` within a dialog box template named `UserDialog`. Notice that the dialog box template precedes the statement that creates the dialog record and that the `Dialog` statement follows both of them in the script.

```
Sub Main()
    Dim ListBox1$()           'Initialize listbox array
    'Define the dialog box template
    Begin Dialog UserDialog , ,163,94,"Grocery Order"
        Text 13,6,32,8,"&Quantity:",.Text1
        TextBox 48,4,28,12,.TextBox1
        ListBox 12,28,68,32,ListBox1$,.ListBox1
        OKButton 112,8,40,14
        CancelButton 112,28,40,14
    End Dialog
    Dim b as UserDialog        'Create the dialog record
    Dialog b                   'Display the dialog
End Sub
```

Put information into the custom dialog box

If you open and run the sample script shown in the previous section, you'll see a dialog box that resembles the following (the dialog box you see may be slightly different):



As you can see, this isn't a very useful dialog box. For one thing, the user doesn't see any items in the list box along the left side of this dialog box. To put information into this custom dialog box, you assign values to dialog box controls by modifying the statements in your script that are responsible for displaying those controls to the user. The following table lists the dialog box controls to which you can assign variables and the types of information you can control:

Control(s)	Types of information:
List box, drop-down list box, or combo box	Items
Text box	Default text
Check box	Values

In the following sections, you'll learn how to define and fill an array, set the default text in a text box, and set the initial focus and tab order for the controls in your custom dialog.

Defining and filling an array

You can store items in the list box shown in the example above by creating an array and then assigning values to the elements of the array. For example, you could include the following lines to initialize an array with three elements and assign the names of three common fruits to these elements of your array:

```
Dim Listbox1$(3)                'Initialize listbox array
Listbox1$(0) = "Apples"
Listbox1$(1) = "Oranges"
Listbox1$(2) = "Pears"
```

Setting default text in a text box

You can set the default value of the text box in your script to 12 with the following statement, which must follow the statement that defines the dialog record but precede the statement or function that displays the custom dialog box:

```
b.TextBox1 = "12"
```

Setting the initial focus and controlling the tab order

You can control which control has the focus when your custom dialog box is first displayed as well as the tabbing order between controls by understanding two rules that WM Basic

follows. First, the focus in a custom dialog box is always set initially to the first control to appear in the dialog box template. Second, the order in which subsequent controls appear within the dialog box template determines the tabbing order. That is, pressing the TAB key will change the focus from the first control to the second one, pressing the TAB key again will change the focus to the third control, and so on.

Displaying the Dialog Box

To display a custom dialog box, you can use either a `Dialog()` function or a `Dialog` statement.

Using the `Dialog()` function

You can use a `Dialog()` function to determine how the user closed your custom dialog box. For example, the following statement will return a value when the user clicks on an OK button, Cancel button, or takes another action.

```
response% = Dialog(b)
```

The `Dialog()` function returns any of the following values:

Value returned:	If:
-1	The OK button was clicked.
0	The Cancel button was clicked.
>0	A push button was clicked. The returned number represents which button was clicked based on its order in the dialog box template (1 is the first push button, 2 is the second push button, and so on).

Using the `Dialog` statement

You can use the `Dialog` statement when you don't need to determine how the user closed your dialog box. You'll still be able to retrieve other information from the dialog box record, such as the value of a list box or other dialog box control. The following statement is an example of the correct use of the `Dialog` statement.

```
Dialog b
```

Retrieving values from the Dialog Box

After displaying a custom dialog box for your user, your script must retrieve the values of the dialog controls. You retrieve these values by referencing the appropriate identifiers in the dialog record.

You'll find an example of how to retrieve values from a custom dialog box in the following sample script.

Sample

In the following script, named `DIALOG2.WBS`, shows several of the techniques described earlier in this section have been used.

In this script, the array named `ListBox1` is filled with three elements ("Apples", "Oranges", and "Pears"). The default value of `TextBox1` is set to 12. A variable named `response` is used to store information about how the dialog box was closed. An identifier named `ListBox1` is used to determine whether the user chose "Apples", "Oranges", or "Pears" in the list box

named `Listbox$`. Finally, a `Select Case...End Select` statement is used to display a message box appropriate to the manner in which the user dismissed the custom dialog box.

```
Sub Main()
    Dim Listbox1$(2)
    Dim response%
    Listbox1$(0) = "Apples"
    Listbox1$(1) = "Oranges"
    Listbox1$(2) = "Pears"
    Begin Dialog UserDialog , ,163,94,"Grocery Order"
        Text 13,6,32,8,"&Quantity:",.Text1
        TextBox 48,4,28,12,.TextBox1
        ListBox 12,28,68,32,Listbox1$,.ListBox1
        OKButton 112,8,40,14
        CancelButton 112,28,40,14
    End Dialog
    Dim b as UserDialog
    b.TextBox1 = "12"
    response = Dialog(b)
    Select Case response%
        Case -1
            Fruit$ = Listbox1$(b.listbox1)
            MsgBox "Thank you for ordering " + b.TextBox1 + " " +
Fruit$
        Case 0
            MsgBox "Your order has been cancelled."
    End Select
End Sub
```

Using a Dynamic Dialog Box

In the previous section, you learned how to use custom dialog boxes. As you learned, you can retrieve the values from dialog box controls after the user dismisses the dialog box by referencing the identifiers in the dialog record.

You can also retrieve values from the dialog box while the dialog box is displayed, using a feature of WM Basic called *dynamic dialog boxes*.

The following script, named `DIALOG3.WBS`, illustrates the most important concepts you'll need to understand in order to create a dynamic dialog box in your script:

```
Dim Fruits(2) as String
Dim Vegetables(2) as String
Function DialogControl(ctrl$, action%, suppvalue%) As Integer
    Select Case action%
        Case 1
            DlgListBoxArray "listbox1", fruits
            DlgValue "listbox1", 0
        Case 2
            If ctrl$ = "OptionButton1" Then
                DlgListBoxArray "listbox1", fruits
                DlgValue "listbox1", 0
            ElseIf ctrl$ = "OptionButton2" Then
                DlgListBoxArray "listbox1", vegetables
                DlgValue "listbox1", 0
            End If
        End Select
    End Function
```

```
Sub Main()  
    Dim ListBox1$()  
    Dim Produce$  
    Fruits(0) = "Apples"  
    Fruits(1) = "Oranges"  
    Fruits(2) = "Pears"  
    Vegetables(0) = "Carrots"  
    Vegetables(1) = "Peas"  
    Vegetables(2) = "Lettuce"  
    Begin Dialog UserDialog , ,163,94,"Grocery  
Order",.DialogControl  
        Text 13,6,32,8,"&Quantity:",.Text1  
        TextBox 48,4,28,12,.TextBox1  
        ListBox 12,28,68,32,ListBox1$,.ListBox1  
        OptionGroup .OptionGroup1  
            OptionButton 12,68,48,8,"&Fruit",.OptionButton1  
            OptionButton 12,80,48,8,"&Vegetables",.OptionButton2  
        OKButton 112,8,40,14  
        CancelButton 112,28,40,14  
    End Dialog  
    Dim b as UserDialog  
    b.TextBox1 = "12"  
    response% = Dialog(b)  
    Select Case response%  
        Case -1  
            If b.optiongroup1 = 0 Then  
                produce$ = fruits(b.listbox1)  
            Else  
                produce$ = vegetables(b.listbox1)  
            End If  
            MsgBox "Thank you for ordering " & b.TextBox1 & " " &  
produce$  
        Case 0  
            MsgBox "Your order has been cancelled."  
    End Select  
End Sub
```

In the remainder of this section, you'll learn how to make a dialog box dynamic by examining the workings of this sample script.

Making a Dialog Box Dynamic

The first thing to notice about this script, which is a more complex variation of the DIALOG2.WBS script described earlier in this chapter, is that an identifier named `.DialogControl` has been added to the `Begin Dialog` statement. As you will learn in the following section, this parameter to the `Begin Dialog` statement tells WM Basic to pass control to a function procedure named `DialogControl`.

Using a Dialog Function

Before WM Basic displays a custom dialog by executing a `Dialog` statement or `Dialog()` function, it must first initialize the dialog box. During this initialization process, WM Basic checks to see if you've defined a dialog function as part of your dialog box template. If so,

WM Basic will give control to your dialog function, allowing your script to carry out certain actions, such as hiding or disabling dialog box controls.

After completing its initialization process, WM Basic displays your custom dialog box. When the user selects an item in a list box, clears a check box, or carries out certain other actions within the dialog box, WM Basic will again call your dialog function.

In fact, WM Basic also calls your dialog function repeatedly even while the user is not interacting with the dialog box.

Responding to User Actions

A WM Basic dialog function can respond to six types of user actions:

Action	Description
1	This action is sent immediately before the dialog box is shown for the first time.
2	<p>This action is sent when:</p> <ul style="list-style-type: none"> ▪ A button is clicked, such as OK, Cancel, or a push button. ▪ A check box's state has been modified. ▪ An option button is selected. In this case, <i>ControlName\$</i> contains the name of the option button that was clicked, and <i>SuppValue</i> contains the index of the option button within the option button group (0-based). ▪ The current selection is changed in a list box, drop list box, or combo box. In this case, <i>ControlName\$</i> contains the name of the list box, combo box, or drop list box, and <i>SuppValue</i> contains the index of the new item (0 is the first item, 1 is the second, and so on).
3	This action is sent when the content of a text box or combo box has been changed. This action is only sent when the control loses focus.
4	This action is sent when a control gains the focus.
5	This action is sent continuously when the dialog box is idle.
6	This action is sent when the dialog box is moved.

You'll find a more complete explanation of these action codes in *Chapter 2, A-Z Reference* in the *Working Model Basic User's Manual* in the reference for `DlgProc` function.

Controlling Working Model from Another Application

You can use another application to control Working Model through DDE (on Windows) or Apple events (on Macintosh). You can write a script or a macro using the external application, that sends instructions to Working Model in WM Basic language. This chapter provides instructions to establish the communication between applications.

Contents

- Sending WM Basic Program via DDE
- Sending WM Basic Program via Apple events

Sending a WM Basic Program via DDE

Working Model is a DDE server and supports the topic `WMBasic`. Via DDE, any application can send WM Basic commands to Working Model. Shown below are a few examples of how you can send instructions to Working Model from another application.

Note: The method for establishing a DDE conversation varies significantly from one application to another. Please refer to the documentation for the application for reference.

Using Microsoft Excel 5.0

You can write a Visual Basic macro in Excel that directly sends a series of instructions in WM Basic to Working Model. Please refer to the User's Manual for Excel to find out more details.

Sending a Single Line of WM Basic Code

Shown below is a Visual Basic script written as an Excel macro.

```
Sub Macro1()  
    ' Assume Working Model is running  
    channelID = Application.DDEInitiate("WM", "WMBasic")  
    Application.DDEExecute channelID, "MsgBox "Hello""  
    Application.DDETerminate channelID  
End Sub
```

- The first line, `DDEInitiate`, establishes the link with WM (Working Model, which runs as `WM.EXE`) under the topic `WMBasic`. Working Model must be running for `DDEInitiate` to work. The function `DDEInitiate` returns the `channelID` (an integer describing the channel number) which will be used in subsequent DDE conversations.
- `DDEExecute` statement takes the first parameter `channelID`, and the program written in WM Basic as the second parameter. Note that the double-quotation marks (") need to be repeated when sent through `DDEExecute`.
- To finish the DDE conversation, add `DDETerminate` statement in your macro.

When Working Model receives the above statements, `Sub Main()` and `End Sub` are automatically inserted to create a complete program as follows:

```
Sub Main()  
    MsgBox "Hello"  
End Sub
```

Sending a WM Basic Program

If you want to send more than one line of WM Basic code,

- the code needs to be a complete program (includes `Sub Main()` and `End Sub`), and
- you need to bracket the beginning and the end of the program by special symbols `$wmstart$` and `$wmend$`. Otherwise, Working Model would add `Sub Main()` and `End Sub` to every line, and your program would not function properly.

For example:

```
Sub Macro1()
    channelID = Application.DDEInitiate("WM", "WMBasic")
    Application.DDEExecute channelID, "$wmstart$"
    Application.DDEExecute channelID, "Sub Main()"
    Application.DDEExecute channelID, "Dim Box as WMBody"
    Application.DDEExecute channelID, "Set Box =
WM.ActiveDocument.NewBody("square")"
    Application.DDEExecute channelID, "Box.Width.Value = 0.2"
    Application.DDEExecute channelID, "End Sub"
    Application.DDEExecute channelID, "$wmend$"
    Application.DDETerminate channelID
End Sub
```

Working Model will start processing the target program upon receiving \$wmend\$. The above code is interpreted as the following WM Basic program:

```
Sub Main()
    Dim Box as WMBody
    Set Box = WM.ActiveDocument.NewBody("square")
    Box.Width.Value = 0.2
End Sub
```

Using Visual Basic

You can write a standalone Visual Basic program that sends a series of instructions to Working Model. This scheme may seem a bit strange at first, since WM Basic closely mimics Visual Basic. After all, what is wrong with writing your entire application in WM Basic?

Nothing, really. But under following circumstances, you may find it appropriate to take advantage of the facility.

- You already have a large application written in Visual Basic, and you wish to modify a part of the program so that it interacts with Working Model.
- You wish to write a Visual Basic application, but most of its functionality does not require Working Model. For instance, you wish to distribute your application to other people who may or may not own Working Model, and the interactions with Working Model are not a critical part of your application.

Sending a Single Line of WM Basic Code

Shown below is an example of how you may send a command from Visual Basic program. Assume the variable `myControl` is a valid Visual Basic control.

```
myControl.LinkTopic = "WM|WMBasic" ' specifies app and topic
myControl.LinkMode = 2
myControl.LinkExecute "MsgBox ""Hello"""
```

(Note that the two repeating quotation marks (" ") are necessary to indicate a single quotation symbol within the parameter of `LinkExecute`.)

The second line (`myControl.LinkMode = 2`) sets the link mode to 2 (Manual Mode).

Working Model only supports the manual mode link. For more information, please refer to the documentation on Visual Basic.

When Working Model receives the above statements, `Sub Main()` and `End Sub` are automatically inserted to create a complete program as follows:

```
Sub Main()
```

```
MsgBox "Hello"  
End Sub
```

Sending a WM Basic Program

If you wish send multiple lines of WM Basic code, you need to call the `LinkExecute` method repeatedly. Furthermore:

- the code needs to be a complete program (includes `Sub Main()` and `End Sub`), and
- you need to bracket the beginning and the end of the program by special symbols `$wmstart$` and `$wmend$`. Otherwise, Working Model would add `Sub Main()` and `End Sub` to every line, and your program would not function properly.

For example:

```
myControl.LinkTopic = "WM|WMBasic" ' specifies app and topic  
myControl.LinkItem = ""  
myControl.LinkMode = 2  
myControl.LinkExecute "$wmstart$"  
myControl.LinkExecute "Sub Main()  
myControl.LinkExecute "Dim B as WMbody"  
myControl.LinkExecute "Set B =  
WM.ActiveDocument.NewBody("square")"  
myControl.LinkExecute "B.Width.Value = 0.2"  
myControl.LinkExecute "End Sub"  
myControl.LinkExecute "$wmend$"
```

Working Model will start processing the code upon receiving `$wmend$`.

Sending a WMBasic Program via Apple Events

Working Model supports the Required suite of Apple events. It also supports the DoScript event (part of Miscellaneous suite). Shown below is an example of how you can send instructions to Working Model from AppleScript Editor. Other applications (although not discussed here) supporting Apple events include Claris FileMaker.

Note: The method for establishing an Apple events communication varies significantly from one application to another. Please refer to the documentation for the application for reference.

Using AppleScript Editor

AppleScript Editor is an editing tool used to create scripts that automate various Macintosh operations. AppleScript Editor is included in Macintosh System 7.5.

You can send a WM Basic program through AppleScript using a DoScript statement.

Sending a Single Line of WM Basic Code

Shown below is an example of an AppleScript code which executes WM Basic code:

```
tell application "Working Model"  
  DoScript "MsgBox \"Hello\""  
end tell
```

Note that the backslash (\) must be added in front of a double-quotation mark (") to be properly interpreted by the DoScript command.

When Working Model receives the above statements, `Sub Main()` and `End Sub` are automatically inserted to create a complete program as follows:


```
Sub Main()  
  MsgBox "Hello"  
End Sub
```

Sending a WM Basic Program

If you wish send multiple lines of WM Basic code, you need to call the `DoScript` command repeatedly. Furthermore:

- the code needs to be a complete program (includes `Sub Main()` and `End Sub`), and
- you need to bracket the beginning and the end of the program by special symbols `$wmstart$` and `$wmend$`. Otherwise, Working Model would add `Sub Main()` and `End Sub` to every line, and your program would not function properly

For example:

```
tell application "Working Model"  
  DoScript "$wmstart$"  
  DoScript "Sub Main()  
  DoScript "Dim B as WMBody"  
  DoScript "Set B = WM.ActiveDocument.NewBody(\"square\")"  
  DoScript "B.Width.Value = 0.2"  
  DoScript "End Sub"  
  DoScript "$wmend$"  
end tell
```

Working Model will start processing the code upon receiving `$wmend$`.

Runtime Error Messages

This section contains listings of all the runtime errors. It is divided into three subsections, the first describing errors messages compatible with "standard" Basic as implemented by Microsoft Visual Basic, the second describing error messages specific to WM Basic, and the third describing error messages that only pertain to Working Model operations. A few error messages contain placeholders which get replaced by the runtime when forming the completed runtime error message. These placeholders appear in the following list as the italicized word *placeholder*.

For details on how you can trap and handle errors please see the section on **Error Handling** (topic) in the Chapter 2 of this manual.

Visual Basic Compatible Error Messages

Error Number	Error Message
3	Return without GoSub
5	Illegal procedure call
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
19	No Resume
20	Resume without error
26	Dialog needs End Dialog or push button

Error Number	Error Message
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
49	Bad DLL calling convention
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
59	Bad record length
61	Disk full
62	Input past end of file
63	Bad record number
64	Bad file name
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set
93	Invalid pattern string
94	Invalid use of Null
139	Only one user dialog may be up at any time
140	Dialog control identifier does not match any current control

Error Number	Error Message
141	The <i>placeholder</i> statement is not available on this dialog control type
143	The dialog control with the focus may not be hidden or disabled
144	Focus may not be set to a hidden or disabled control
150	Dialog control identifier is already defined
163	This statement can only be used when a user dialog is active
260	No timer available
281	No more DDE channels
282	No foreign application responded to a DDE initiate
283	Multiple applications responded to a DDE initiate
285	Foreign application won't perform DDE method or operation
286	Timeout while waiting for DDE response
287	User pressed Escape key during DDE operation
288	Destination is busy
289	Data not provided in DDE operation
290	Data in wrong format
291	Foreign application quit
292	DDE conversation closed or changed
295	Message queue filled; DDE message lost
298	DDE requires ddeml.dll
429	OLE Automation server can't create object
430	Class doesn't support OLE Automation
431	OLE Automation server cannot load file
432	File name or class name not found during OLE Automation operation
433	OLE Automation object does not exist
434	Access to OLE Automation object denied
435	OLE initialization error

Error Number	Error Message
436	OLE Automation method returned unsupported type
437	OLE Automation method did not return a value
438	Object doesn't support this property or method <i>placeholder</i>
439	OLE Automation argument type mismatch <i>placeholder</i>
440	OLE Automation error <i>placeholder</i>
443	OLE Automation Object does not have a default value
452	Invalid ordinal
460	Invalid Clipboard format
520	Can't empty clipboard
521	Can't open clipboard
600	Set value not allowed on collections
601	Get value not allowed on collections
603	ODBC - SQLAllocEnv failure
604	ODBC - SQLAllocConnect failure
608	ODBC - SQLFreeConnect error
610	ODBC - SQLAllocStmt failure
3129	Invalid SQL statement; expected 'DELETE', 'INSERT', 'PROCEDURE', 'SELECT', or 'UPDATE'
3146	ODBC - call failed
3148	ODBC - connection failed
3276	Invalid database ID

WM Basic-Specific Error Messages

Error Number	Error Message
800	Incorrect Windows version
801	Too many dimensions
802	Can't find window
803	Can't find menu item

Error Number	Error Message
804	Another queue is being flushed
805	Can't find control
806	Bad channel number
807	Requested data not available
808	Can't create pop-up menu
809	Message box canceled
810	Command failed
811	Network error
812	Network function not supported
813	Bad password
814	Network access denied
815	Network function busy
816	Queue overflow
817	Too many dialog controls
818	Can't find list box/combo box item
819	Control is disabled
820	Window is disabled
821	Can't write to ini file
822	Can't read from ini file
823	Can't copy file onto itself
824	OLE Automation unknown object name
825	Can't redimension a fixed array
826	Can't load and initialize extension
827	Can't find extension
828	Unsupported function or statement
829	Can't find ODBC libraries
830	OLE Automation Lbound or Ubound on non-Array value
831	Incorrect definition for dialog procedure

Error Messages in Working Model Operations

Error Number	Error Message
900	Failure during Working Model API call
901	A Working Model API call failed because the object it referenced no longer exists
902	Attempted to set a property which the referenced object does not have
903	Attempted to get a property which the referenced object does not have
904	Unrecognized string parameter in subroutine call
905	Bad parameter in subroutine call
906	Could not load the specified library. Check its location and confirm that it compiles.
907	Could not run the specified script. Check its location and confirm that it compiles.
999	Invalid object variable

Compiler Error Messages

The following table contains a list of all the errors generated by the WM Basic compiler. With some errors, the compiler changes placeholders within the error to text from the script being compiled. These placeholders are represented in this table by the word *placeholder*.

1	Variable Required - Can't assign to this expression
2	Letter range must be in ascending order
3	Redefinition of default type
4	Out of memory, too many variables defined
5	Type-character doesn't match defined type
6	Expression too complex
7	Cannot assign whole array
8	Assignment variable and expression are different types
10	Array type mismatch in parameter
11	Array type expected for parameter
12	Array type unexpected for parameter
13	Integer expression expected for an array index
14	Integer expression expected
15	String expression expected
18	Left of "." must be an object, structure, or dialog
19	Invalid string operator
20	Can't apply operator to array type
21	Operator type mismatch
22	" <i>placeholder</i> " is not a variable
23	" <i>placeholder</i> " is not a array variable or a function

- 24 Unknown *placeholder* "*placeholder*"
- 25 Out of memory
- 26 *placeholder*: Too many parameters encountered
- 27 *placeholder*: Missing parameter(s)
- 28 *placeholder*: Type mismatch in parameter *placeholder*
- 29 Missing label "*placeholder*"
- 30 Too many nested statements
- 31 Encountered new-line in string
- 32 Overflow in decimal value
- 33 Overflow in hex value
- 34 Overflow in octal value
- 35 Expression is not constant
- 37 No type-characters allowed on parameters with explicit type
- 39 Can't pass an array by value
- 40 "*placeholder*" is already declared as a parameter
- 41 Variable name used as label name
- 42 Duplicate label
- 43 Not inside a function
- 44 Not inside a sub
- 46 Can't assign to function
- 47 Identifier is already a variable
- 48 Unknown type
- 49 Variable is not an array type
- 50 Can't redimension an array to a different type
- 51 Identifier is not a string array variable
- 52 0 expected
- 55 Integer expression expected for file number
- 56 *placeholder* is not a method of the object
- 57 *placeholder* is not a property of the object

58	Expecting 0 or 1
59	Boolean expression expected
60	Numeric expression expected
61	Numeric type FOR variable expected
62	For...Next variable mismatch
63	Out of string storage space
64	Out of identifier storage space
65	Internal error 1
66	Maximum line length exceeded
67	Internal error 3
68	Division by zero
69	Overflow in expression
70	Floating-point expression expected
72	Invalid floating-point operator
74	Single character expected
75	Subroutine identifier can't have a type-declaration character
76	Script is too large to be compiled
77	Variable type expected
78	Can't evaluate expression
79	Can't assign to user or dialog type variable
80	Maximum string length exceeded
81	Identifier name already in use as another type
84	Operator cannot be used on an object
85	<i>placeholder</i> is not a property or method of the object
86	Type-character not allowed on label
87	Type-character mismatch on routine <i>placeholder</i>
88	Destination name is already a constant
89	Can't assign to constant
90	Error in format of compiler extensions
91	Identifier too long

- 92 Expecting string or structure expression
- 93 Can't assign to expression
- 94 Dialog and Object types are not supported in this context
- 95 Array expression not supported as parameter
- 96 Dialogs, objects, and structures expressions are not supported as a parameter
- 97 Invalid numeric operator
- 98 Invalid structure element name following "."
- 99 Access value can't be used with specified mode
- 101 Invalid operator for object
- 102 Can't LSet a type with a variable-length string
- 103 Syntax error
- 104 *placeholder* is not a method of the object
- 105 No members defined
- 106 Duplicate type member
- 107 Set is for object assignments
- 108 Type-character mismatch on variable
- 109 Bad octal number
- 110 Bad number
- 111 End-of-script encountered in comment
- 112 Misplaced line continuation
- 113 Invalid escape sequence
- 114 Missing End Inline
- 115 Statement expected
- 116 ByRef argument mismatch
- 117 Integer overflow
- 118 Long overflow
- 119 Single overflow
- 120 Double overflow
- 121 Currency overflow
- 122 Optional argument must be Variant

- 123 Parameter must be optional
- 124 Parameter is not optional
- 125 Expected: Lib
- 126 Illegal external function return type
- 127 Illegal function return type
- 128 Variable not defined
- 129 No default property for the object
- 130 The object does not have an assignable default property
- 131 Parameters cannot be fixed length strings
- 132 Invalid length for a fixed length string
- 133 Return type is different from a prior declaration
- 134 Private variable too large. Storage space exceeded
- 135 Public variables too large. Storage space exceeded

A P P E N D I X C

Language Elements by Platform

The following table lists all WM Basic language elements and on which platforms these language elements are supported. A solid square (■) indicates that the element is supported. The blank square (□) indicates that the element is not supported.

Language Element	Windows	Macintosh
&	■	■
'	■	■
()	■	■
*	■	■
+	■	■
-	■	■
/	■	■
<	■	■
<=	■	■
<>	■	■
= (assignment)	■	■
= (operator)	■	■
>	■	■
>=	■	■
\	■	■
^	■	■
_	■	■
Abs	■	■
ActivateControl	■	□

Language Element	Windows	Macintosh
And	■	■
AnswerBox	■	■
AppActivate	■	■
AppClose	■	□
AppFileName\$	■	□
AppFind	■	□
AppGetActive\$	■	□
AppGetPosition	■	□
AppGetState	■	□
AppHide	■	□
AppList	■	□
AppMaximize	■	□
AppMinimize	■	□
AppMove	■	□
AppRestore	■	□
AppSetState	■	□
AppShow	■	□
AppSize	■	□
AppType	■	□
ArrayDims	■	■
ArraySort	■	■
Asc	■	■
AskBox\$	■	■
AskPassword\$	■	■
Atn	■	■
Basic.Capability	■	■
Basic.Eoln\$	■	■
Basic.FreeMemory	■	■
Basic.HomeDir\$	■	■
Basic.OS	■	■
Basic.PathSeparator\$	■	■

Language Element	Windows	Macintosh
Basic.Version\$	■	■
Beep	■	■
Begin Dialog	■	■
Boolean	■	■
ButtonEnabled	■	□
ButtonExists	■	□
Call	■	■
CancelButton	■	■
CBool	■	■
CCur	■	■
CDate	■	■
Cdbl	■	■
ChDir	■	■
ChDrive	■	□
CheckBox	■	■
CheckBoxEnabled	■	□
CheckBoxExists	■	□
Choose	■	■
Chr, Chr\$	■	■
CInt	■	■
Clipboard\$ (function)	■	■
Clipboard\$ (statement)	■	■
Clipboard.Clear	■	■
Clipboard.GetFormat	■	■
Clipboard.GetText	■	■
Clipboard.SetText	■	■
CLng	■	■
Close	■	■
ComboBox	■	■
ComboBoxEnabled	■	□
ComboBoxExists	■	□

Language Element	Windows	Macintosh
Command, Command\$	■	■
Const	■	■
Cos	■	■
CreateObject	■	■
CSng	■	■
CStr	■	■
CurDir, CurDir\$	■	■
Currency	■	■
CVar	■	■
CVDate	■	■
CVErr	■	■
Date	■	■
Date, Date\$ (functions)	■	■
Date, Date\$ (statements)	■	■
DateAdd	■	■
DateDiff	■	■
DatePart	■	■
DateSerial	■	■
DateValue	■	■
Day	■	■
DDB	■	■
DDEExecute	■	□
DDEInitiate	■	□
DDEPoke	■	□
DDERequest, DDERequest\$	■	□
DDESend	■	□
DDETerminate	■	□
DDETerminateAll	■	□
DDETimeOut	■	□
Declare	■	■
DefBool	■	■

Language Element	Windows	Macintosh
DefCur	■	■
DefDate	■	■
DefDbl	■	■
DefInt	■	■
DefLng	■	■
DefObj	■	■
DefSng	■	■
DefStr	■	■
DefVar	■	■
Desktop.ArrangeIcons	■	□
Desktop.Cascade	■	□
Desktop.SetColors	■	□
Desktop.SetWallpaper	■	□
Desktop.Snapshot	■	□
Desktop.Tile	■	□
Dialog (function)	■	■
Dialog (statement)	■	■
Dim	■	■
Dir, Dir\$	■	■
DiskDrives	■	□
DiskFree	■	□
DlgControlId	■	■
DlgEnable (function)	■	■
DlgEnable (statement)	■	■
DlgFocus (function)	■	■
DlgFocus (statement)	■	■
DlgListBoxArray (function)	■	■
DlgListBoxArray (statement)	■	■
DlgSetPicture	■	■
DlgText (statement)	■	■
DlgText\$ (function)	■	■

Language Element	Windows	Macintosh
DlgValue (function)	■	■
DlgValue (statement)	■	■
DlgVisible (function)	■	■
DlgVisible (statement)	■	■
Do...Loop	■	■
DoEvents (function)	■	■
DoEvents (statement)	■	■
DoKeys	■	□
Double	■	■
DropListBox	■	■
EditEnabled	■	□
EditExists	■	□
End	■	■
Environm Environ\$	■	■
Eof	■	■
Eqv	■	■
Erase	■	■
Erl	■	■
Err (function)	■	■
Err (statement)	■	■
Error	■	■
Error, Error\$	■	■
Exit Do	■	■
Exit For	■	■
Exit Function	■	■
Exit Sub	■	■
Exp	■	■
FileAttr	■	■
FileCopy	■	■
FileDateTime	■	■
FileDirs	■	■

Language Element	Windows	Macintosh
FileExists	■	■
FileLen	■	■
FileList	■	■
FileParse\$	■	■
FileType	■	□
Fix	■	■
For...Next	■	■
Format, Format\$	■	■
FreeFile	■	■
Function...End	■	■
Fv	■	■
Get	■	■
GetAttr	■	■
GetCheckBox	■	□
GetComboBoxItem\$	■	□
GetComboBoxItemCount	■	□
GetEditText\$	■	□
GetListBoxItem\$	■	□
GetListBoxItemCount	■	□
GetObject	■	■
GetOption	■	□
Global	■	■
GoSub	■	■
Goto	■	■
GroupBox	■	■
Hex, Hex\$	■	■
HLine	■	□
Hour	■	■
HPage	■	□
HScroll	■	□
HWND	■	□

Language Element	Windows	Macintosh
HWND.Value	■	□
If...Then...Else	■	■
IIf	■	■
Imp	■	■
Inline	■	■
Input#	■	■
Input, Input\$	■	■
InputBox, InputBox\$	■	■
InStr	■	■
Int	■	■
Integer	■	■
IPmt	■	■
IRR	■	■
Is	■	■
IsDate	■	■
IsEmpty	■	■
IsError	■	■
IsMissing	■	■
IsNull	■	■
IsNumeric	■	■
IsObject	■	■
Item\$	■	■
ItemCount	■	■
Kill	■	■
LBound	■	■
LCase, LCase\$	■	■
Left, Left\$	■	■
Len	■	■
Let	■	■
Like	■	■
Line Input #	■	■

Language Element	Windows	Macintosh
Line\$	■	■
LineCount	■	■
ListBox	■	■
ListBoxEnabled	■	□
ListBoxExists	■	□
Loc	■	■
Lock	■	■
Lof	■	■
Log	■	■
Long	■	■
LSet	■	■
LTrim, LTrim\$	■	■
MacID	□	■
MacScript	□	■
Main	■	■
Mci	■	□
Menu	■	□
MenuItemChecked	■	□
MenuItemEnabled	■	□
MenuItemExists	■	□
Mid, Mid\$	■	■
Mid, Mid\$	■	■
Minute	■	■
MIRR	■	■
MkDir	■	■
Mod	■	■
Month	■	■
MsgBox (function)	■	■
MsgBox (statement)	■	■
MsgClose	■	□
MsgOpen	■	□

Language Element	Windows	Macintosh
MsgSetText	■	□
MsgSetThermometer	■	□
Name	■	■
Net.AddCon\$	■	□
Net.Browse\$	■	□
Net.CancelCon	■	□
Net.Dialog	■	□
Net.GetCaps	■	□
Net.GetCon\$	■	□
Net.User\$	■	□
Not	■	■
Nothing	■	■
Now	■	■
NPer	■	■
Npv	■	■
Object	■	■
Oct. Oct\$	■	■
OKButton	■	■
On Error	■	■
Open	■	■
OpenFilename\$	■	■
Option Base	■	■
Option Compare	■	■
Option CStrings	■	■
OptionButton	■	■
OptionEnabled	■	□
OptionExists	■	□
OptionGroup	■	■
Or	■	■
Picture	■	■
PictureButton	■	■

Language Element	Windows	Macintosh
Pmt	■	■
PopupMenu	■	□
PPmt	■	■
Print	■	■
Print #	■	■
PrinterGetOrientation	■	□
PrinterSetOrientation	■	□
PrintFile	■	□
Private	■	■
Public	■	■
PushButton	■	■
Put	■	■
Pv	■	■
QueEmpty	■	□
QueFlush	■	□
QueKeyDn	■	□
QueKeys	■	□
QueKeyUp	■	□
QueMouseClicked	■	□
QueMouseDblClk	■	□
QueMouseDblDn	■	□
QueMouseDn	■	□
QueMouseMove	■	□
QueMouseMoveBatch	■	□
QueMouseUp	■	□
QueSetRelativeWindow	■	□
Random	■	■
Randomize	■	■
Rate	■	■
ReadINI\$	■	□
ReadINISection	■	□

Language Element	Windows	Macintosh
ReDim	■	■
REM	■	■
Reset	■	■
Resume	■	■
Return	■	■
Right, Right\$	■	■
Rmdir	■	■
Rnd	■	■
RSet	■	■
RTrim, RTrim\$	■	■
SaveFileName\$	■	■
Screen.DlgBaseUnitsX	■	□
Screen.DlgBaseUnitsY	■	□
Screen.Height	■	□
Screen.TwipsPerPixelX	■	□
Screen.TwipsPerPixelY	■	□
Screen.Width	■	□
Second	■	■
Seek	■	■
Seek	■	■
Select...Case	■	■
SelectBox	■	■
SelectButton	■	□
SelectComboboxItem	■	□
SelectListBoxItem	■	□
SendKeys	■	□
Set	■	■
SetAttr	■	■
SetCheckbox	■	□
SetEditText	■	□
SetOption	■	□

Language Element	Windows	Macintosh
Sgn	■	■
Shell	■	■
Sin	■	■
Single	■	■
Sleep	■	■
Sln	■	■
Space, Space\$	■	■
Spc	■	■
SQLBind	■	□
SQLClose	■	□
SQLError	■	□
SQLExecQuery	■	□
SQLGetSchema	■	□
SQLOpen	■	□
SQLRequest	■	□
SQLRetrieve	■	□
SQLRetrieveToFile	■	□
Sqr	■	■
Stop	■	■
Str, Str\$	■	■
StrComp	■	■
String	■	■
String, String\$	■	■
Sub...End	■	■
Switch	■	■
SYD	■	■
System.Exit	■	□
System.FreeMemory	■	□
System.FreeResources	■	□
System.MouseTrails	■	□
System.Restart	■	□

Language Element	Windows	Macintosh
System.TotalMemory	■	□
System.WindowsDirectory\$	■	□
System.WindowsVersion\$	■	□
Tab	■	■
Tan	■	■
Text	■	■
TextBox	■	■
Time, Time\$ (functions)	■	■
Time, Time\$ (statements)	■	■
Timer	■	■
TimeSerial	■	■
TimeValue	■	■
Trim, Trim\$	■	■
Type	■	■
UBound	■	■
UCase, UCase\$	■	■
UnLock	■	■
Val	■	■
Variant	■	■
VarType	■	■
ViewportClear	■	□
ViewportClose	■	□
ViewportOpen	■	□
VLine	■	□
VPage	■	□
VScroll	■	□
Weekday	■	■
While...Wend	■	■
Width#	■	■
WinActivate	■	□
WinClose	■	□

Language Element	Windows	Macintosh
WinFind	■	□
WinList	■	□
WinMaximize	■	□
WinMinimize	■	□
WinMove	■	□
WinRestore	■	□
WinSize	■	□
Word\$	■	■
WordCount	■	■
Write #	■	■
WriteINI	■	□
Xor	■	■
Year	■	■

WM Basic Limitations

The following list contains important WM Basic limitations:

- Line numbers are not supported. Labels can be used in place of line numbers are targets for the `Goto` statement.
- Strings are limited in length to 32,764 characters. This includes local, public, and private strings, as well as strings within structures and arrays.
- The Visual Basic declaration modifiers `Static` and `Shared` are not supported.
- The default string space is 8K, which expands automatically up to a maximum of 1 MB. This space contains all strings and arrays regardless of their scope.
- The default stack size for the runtime is 2,048 bytes. This space contains all local variables (except arrays and variable-length strings) and passed parameters.
The stack is also used by the runtime for storage of intermediate values, so the actual available stack space will be slightly less.
Calls made to subroutines or functions in other scripts use the stack of the caller.
- The data area that holds private variables is limited to 16K. This data space contains all private variables except strings and arrays, which are stored in the string space.
- The data area that holds public variables is limited to 16K. This data space contains all public variables except strings and arrays, which are stored in the string space.
- The size of a source script is limited to 65534 characters. This limitation can be avoided by breaking up large scripts into smaller ones.
- A compiled script consists of p-code, constant data, and symbolic information, each of which is limited to 64K. These limitations can be avoided by breaking up large scripts into smaller ones, which is rarely necessary.
- Arrays can have up to 60 dimensions.
- Variable names are limited to 80 characters.

- Labels are limited to 80 characters.
- Each executing script contains a table of structures that track calls made to external routines. Each structure is approximately 88 bytes with an overall size limited to 64K.
- The number of open DDE channels is not fixed; rather, it is limited only by available memory and system resources.
- The number of open files is limited to 255 or the operating system limit, whichever is less.
- The number of characters within a string literal (a string enclosed within quotation marks) is limited to 1024 characters. (Strings can be concatenated using the concatenation [&] operator with the normal string limit of 32,764 characters.)
- The number of nesting levels (i.e., loops within loops) is limited by compiler memory.
- Queue playback buffer size is limited to 64K. With 10 bytes per event, this allows for 6,553 events.
- Each GoSub requires 2 bytes of the WM Basic runtime stack.
- Arrays and user-defined types cannot be passed to a method of an OLE automation object.
- Arrays and user-defined types cannot be set as the value of a property of an OLE automation object.
- Arrays and user-defined types cannot be returned from a method or property of an OLE automation object.
- Array indexes must be in the following range:
$$-32768 \leq \text{array-index} \leq 32767$$
- The size of an array cannot exceed 32K. For example, an array of integers, each of which requires 2 bytes of storage, is limited to the following maximum number of elements:
$$\text{max_num_elements} = (32767 - \text{overhead}) / 2$$

where *overhead* is currently approximately 16 bytes.
A maximum of 128 fonts can be used within a single user dialog, although the practical limitation imposed by the operating system may be less.

A P P E N D I X E

WM Basic/Visual Basic Differences

The following section describes differences between Visual Basic and WM Basic. In the proceeding discussion, "VB" is used to refer to Visual Basic 3.0, and VBA is used to refer to Visual Basic for Applications 1.0.

The following sections are covered:

- Arrays
- Constants
- Data Types
- Declarations
- Declare Statement
- Floating Point Numbers
- Language Element Differences
- Natural Language Support
- Objects
- OLE Automation
- Parameter Passing
- Strings
- Variants
- Stack Size
- Expression Evaluation
- File Searching

Arrays

VB and VBA support huge arrays, WM Basic does not.

Constants

VBA supports shared constants (using the `Public` keyword). In WM Basic, constants must be repeated within each script in which they are used.

VB and VBA do not allow the concatenation of constant elements. For example, the following script compiles in WM Basic but not in VB or VBA:

```
Const t$ = "Hello" & Chr$(9) & "there."  
Sub Main()  
    MsgBox t$  
End Sub
```

VBA allows a user to redefine global constants at the subroutine/function level without affecting their global values; WM Basic does not. For example, the following script will compile and execute in VBA but not in WM Basic:

```
Const t = "Hello"  
Sub Main()  
    Const t$ = "Good Bye"  
    MsgBox t$           'Displays Good Bye  
End Sub
```

Data Types

WM Basic and VBA support the `Boolean` and `Date` data types, VB does not.

Declarations

In VB and VBA, if a variable is initially declared with a type declaration character, then that character must appear with every use of that variable. WM Basic relaxes this by not requiring the type declaration character with every use of that variable.

Both VB and VBA support the `Static` keyword as a modifier for the `Sub` and `Function` statements. WM Basic supports use of this keyword with these statements with no effect.

A variable used in a comparison expression that hasn't been declared will be implicitly declared in VB and VBA. In WM Basic, this will be seen as an unresolved function:

```
Sub Main  
    if a$ = "hello" then beep  
End Sub
```

In WM Basic, the above script will compile, but gives a `Sub or function not defined` error when executed. In VB and VBA, this will automatically declare a variable called `a$` as a `String`.

WM Basic allows the `@` type declaration character to be specified with currency constants; VB and VBA do not.

Declare Statement

VBA supports shared `Declare` statements (using the `Public` keyword). In WM Basic, these must be declared in every script in which they are used.

WM Basic supports a superset of that functionality available in VB—namely, the additional calling conventions.

WM Basic and VB pass values to external routines in the same manner with the following exceptions:

- WM Basic passes `True` or `False` as Boolean values (signed short in C). VB passes these as Boolean variants.
- Variants are passed as internal variant structures in both WM Basic and VB. For all numeric values, the types are the same. Strings, however, in WM Basic are passed as a 16-bit internal value, whereas in VB they are passed as a 32-bit pointer to a null-terminated string. The variant structure in both systems is a 4-byte type (a 32-bit integer—the same value as returned by the `VarType` function), followed by 4 bytes of slop, followed by the value of the variable, as shown below:

Bytes 0-3	Bytes 4-7	Bytes 8-15
VarType	Alignment slop	Value

- Strings within variants are passed within an internal variant structure in both WM Basic and VB.

Floating Point Numbers

In VB and VBA, floating point numbers are interpreted as doubles unless they are explicitly accompanied by a type-declaration character. Thus, the following line assigns a `Double` in VB and VBA, whereas in WM Basic, it assigns a `Single`:

```
a = 0.00001
```

In WM Basic, additional checking is performed to determine if a floating point number can be accurately represented as a `Single`. If so, then the number is stored as a `Single`, requiring 4 bytes rather than 8.

The implications of this difference can be seen in the following code:

```
Dim a As Variant, b As Variant
a = 1000
b = .00001
a = a + b
MsgBox a
```

In VB and VBA, since the variables `a` and `b` are assigned `Double` values, the addition is performed between two doubles, resulting in the value `1000.00001`. In WM Basic, on the other hand, `a` and `b` are assigned `Single` values, resulting in an addition between two singles. When these two singles are added, there is a loss of precision resulting in the value `1000`. In situations such as this, you should explicitly force the types using type-declaration characters. The above code can be re-written as follows:

```
Dim a As Variant, b As Variant
a = 1000#
b = .00001#
a = a + b
MsgBox a      'WM Basic displays 1000.00001
```

Currency Numbers

In VB, `Double` numbers do not convert to `Currency` numbers the same way. In VB, for example, the following script will fail:

```
Sub Main
    result = CCur("-1.401298E-45")
End Sub
```

The above fails in VB because the number being converted is known to be a `Double`. In WM Basic, any number between the valid range supported by `Currency` is convertible to `Currency`, even if the number is expressed in scientific notation or is extremely small (approaching zero).

Language Element Differences

The following language elements allow specification of additional parameters for displaying help in VBA:

```
MsgBox (statement)
MsgBox (function)
InputBox/InputBox$ (functions)
```

WM Basic and VB do not support these parameters.

VBA and WM Basic uses a slightly different syntax for the following SQL functions (due to WM Basic's lack of support for variant arrays):

```
SQLError
SQLGetSchema
SQLRetrieve
SQLRequest
```

The above functions are supported only by VBA, not by VB

WM Basic does not support any of the following VBA language elements:

Language Element	Type
Array	Function
ebHiragana	Constant
ebKatakana	Constant
ebLowerCase	Constant
ebNarrow	Constant
ebProperCase	Constant
ebUpperCase	Constant
ebWide	Constant
Exit Property	Statement
For Each...Next	Statement
IsArray	Function
LenB	Function
LoadPicture	Function
On...Gosub	Statement
On...Goto	Statement
Option Explicit	Statement
Option Private	Statement
Property Get...End Property	Statement
Property Let...End Property	Statement
Property Set...End Property	Statement

Language Element	Type
SavePicture	Statement
Screen.MousePointer	Property
Static	Statement
StrConv	Function
TypeName	Function
TypeOf	Function
With...End With	Statement

The syntax for `MsgBox` and `InputBox` does not support the *context* and *HelpFile* parameters.

Natural Language Support

VBA supports multi-byte characters (sometimes referred to as natural language support); WM Basic and VB do not.

Objects

WM Basic does not support any of VB's objects (except clipboard, screen, and a few others).

OLE Automation

WM Basic does not support named parameters. Visual Basic does not support named-parameters either; this is a feature of VBA.

WM Basic does not support the VBA bracket syntax used with OLE automation objects. For example, the following two expressions are equivalent in VBA:

```
Application.Workbooks(1).Worksheets("Sheet1").Range("A1")
[A1]
```

WM Basic does not support the VBA bracket syntax used to resolve the scope of a method or property:

```
Dim a As Object
Set a = CreateObject("Word.Basic")
a.[MsgBox] "Hello, world." ' <-- Won't work in WM Basic
```

Parameter Passing

VB and WM Basic do not support optional parameters. This is supported by VBA.

Strings

In WM Basic, variable-length strings within structures require 2 bytes of storage. In VB and VBA, variable-length strings within structures require 4 bytes of storage.

The implications of this difference can be seen in the following code:

```
Type Sample
    LastName As String
End Type
Sub Main
    Dim a As Sample
    MsgBox Len(a)
End Sub
```

In the above code, Visual Basic displays 4, while WM Basic displays 2.

In WM Basic, variable-length strings are limited to 32K in length. In VB, variable-length strings are limited to 64K. In VBA, variable-length strings have no limits on their lengths. This limitation has implications on the string functions. For example, the `Left` function returns a specified number of characters from the left side of a string. The second parameter represents the number of characters to return—in VBA, this parameter is a `Long` whereas, in WM Basic, this parameter is an `Integer`. To demonstrate, the following statement will overflow in WM Basic, but not in VBA:

```
s$ = Left(s$,300000)
```

VB and VBA do not accept strings in some functions expecting numbers such as `Int` and `Fix`. WM Basic allows strings as long as they are convertible to numbers.

```
Dim A As Variant
ABS(19)           'OK
A = "10"
ABS(A)           'OK
ABS("10")       'Works in WM Basic, not in VB/VBA
```

In WM Basic, these functions will accept any data type convertible to a number. If the data type is a string, WM Basic converts it to a `Double`.

Fixed-length strings within structures are size-adjusted upwards to an even size. Thus, structures in WM Basic are always even sized. VB and VBA allow fixed-length strings within structures to maintain an odd size.

Variants

Passing variants either by value or by reference to external routines (using the `Declare` statement) passes either the entire variant structure (`ByVal`) or a pointer to a variant structure (`ByRef`) used internally by WM Basic. This means that passing variants to externally declared routines can only be done if that routine is aware of the internal variant structure used by WM Basic. This applies specifically to strings and OLE automation objects stored within the variant.

In VB and VBA, on the other hand, strings and OLE automation objects within variants are stored in their native format (i.e., 32-bit pointer to a null-terminated string or an `LPDISPATCH` value).

VBA supports variant arrays; WM Basic and VB do not. This includes use of the `Array` and `IsArray` functions.

WM Basic and VBA support error variants; VB does not.

Passing Variants by Reference

In VBA, variants cannot be passed by reference to user-defined routines accepting non-variant parameters. For example, the following will not work in VBA:

```
Sub Test(ByRef a As Integer)
End Sub
Sub Main
  Dim v As Variant
  v = 5
  Test v           '<-- VBA gives error here
End Sub
```

In WM Basic, the above example works as expected. WM Basic actually performs a conversion of the `Variant v` to a temporary `Integer` value and passes this temporary value by reference. Upon return from the call to `Test`, WM Basic converts the temporary `Integer` back to a `Variant`.

Passing Optional Variants to Forward Declared Routines

WM Basic does not catch the following error:

```

Declare Sub Test(Optional v As Variant)      '<-- LINE 1
Sub Main
    Test
End Sub
Sub Test(v As Variant)                        '<-- LINE 5
End Sub

```

In the above script, the `Declare` statement on line 1 defines a prototype for the `Test` function that is incompatible with the actual declaration on line 5.

Stack Size

WM Basic uses a default stack of 2K, expandable to 8K. VB and VBA uses a much larger stack size. For example, VBA allocates approximately 14K for the stack. Since the stack for WM Basic is smaller, you may have to be more attentive when using local variables, especially fixed-length strings and structures, since storage for all local variables comes from the stack.

Note: Variable-length strings only require 2 bytes of storage on the stack. Wherever possible, use variable length strings in place of fixed-length strings.

Expression Evaluation

With Boolean expressions (i.e., expression involving `And`, `Or`, `Xor`, `Eqv`, and `Imp`), if one operand is `Null` and the other argument is numeric, then `Null` is returned regardless of the value of the other operand. For example, the following expression returns `Null`:

```
Null And 300000
```

Despite the fact that the expression returns `Null`, VBA evaluates the numeric operand anyway, converting it to a `Long`. If an overflow occurs during conversion, a trappable runtime error is generated. In WM Basic, the expression returns `Null` regardless of the value of the numeric operand. For example, the following expression will overflow in VBA, but not in WM Basic:

```
Null And 5E200
```

File Searching

The filename matching algorithm used by WM Basic is different than that used by VB. This affects commands that perform directory searching, such as `Dir`, `Kill`, and `FileList`. The following differences exist:

- In VB, an asterisk within the filename matches any characters up to the end of the filename or to the period, whichever comes first.
- In VB, the period is a separator for the filename and the extension. In WM Basic, the period is treated as a normal filename character.

The following table describes the meaning of some common file specifications.

Specification	Meaning in VB	Meaning in WM Basic
*	All files.	All files.
.	All files.	All files that have an

		extension.
s*e	All files that begin with "s".	All files that begin with "s" and end with "e".
s*.*	All files that begin with "s".	All files that begin "s" and have an extension.
test.	The file "test" with no extension.	The file called "test.". WM Basic will never find this file under Windows or DOS.
test.*	All files having the root name "test" with any extension, such as "test", "test.txt", and so on.	All files having the root name "test" with an extension. The file "test" with no extension will not be found.

This filename matching algorithm is the same across all platforms that support WM Basic.

Index

- (minus sign), subtraction operator, 52–53
- ! (exclamation point)
 - activating parts of files, 260
 - used for **Single** type-declaration character. *See* type-declaration characters
 - used within user-defined formats, 242
- " (quote), embedding within strings, 305
- # (number sign)
 - as delimiter for date literals, 305
 - delimiter for date literals, 129
 - delimiter for parsing input, 274–76
 - used for **Double** type-declaration character. *See* type-declaration characters
 - used to specify ordinal values, 152
 - used within user-defined formats, 241
 - wildcard used with **Like** (operator), 297
- #ERROR code#**
 - reading from sequential files, 275
 - writing to sequential files, 492
- #FALSE#**
 - reading from sequential files, 275
 - writing to sequential files, 492
- #NULL#**
 - reading from sequential files, 275
 - writing to sequential files, 492
- #TRUE#**
 - reading from sequential files, 275
 - writing to sequential files, 492
- \$ (dollar sign), used for **String** type-declaration character. *See* type-declaration characters
- \$wmsend\$, 695, 696, 698
- \$wmstart\$, 695, 696, 698
- % (percent)
 - used for **Integer** type-declaration character. *See* type-declaration characters
 - used within user-defined formats, 241
- & (ampersand)
 - concatenation operator, 48
 - octal/hexadecimal formats, 305
 - used for **Long** type-declaration character. *See* type-declaration characters
 - used within user-defined formats, 242
- & (operator), vs. addition, 51
- ' (apostrophe), used with comments, 48–50
- () (parentheses)
 - used in expressions, 49–50
- () (parentheses)
 - used to pass parameters by value, 49
- * (asterisk)
 - multiplication operator, 50
 - used within user-defined formats, 242
 - wildcard. *See* wildcards
 - wildcard used with **Like** (operator), 297
- + (plus sign), addition operator, 50–52
- , (comma)
 - used with **Print**, 371
 - used within user-defined formats, 241
- . (period)
 - used to separate object from property, 53–54
 - used with structures, 53–54
 - used within filenames, 124
 - used within user-defined formats, 241
- / (slash)
 - division operator, 54–55
 - used within filenames, 123
 - used within user-defined formats, 241
- : (colon)
 - used with labels, 264
 - used within filenames, 123, 124
 - used within user-defined formats, 241
- ; (semicolon), used with **Print**, 371, 373
- < (less than)
 - comparison operator, 3–4
 - used within user-defined formats, 242

- `<=` (less than or equal), comparison operator, 3–4
- `<>` (not equal), comparison operator, 3–4
- `=` (equal sign)
 - assignment statement, 55–56
 - comparison operator, 3–4
- `>` (greater than)
 - comparison operator, 3–4
 - used within user-defined formats, 242
- `>=` (greater than or equal), comparison operator, 3–4
- `?` (question mark)
 - wildcard. *See* wildcards
 - wildcard used with **Like** (operator), 297
- `@` (at sign)
 - used for **Currency** type-declaration character. *See* type-declaration characters
 - used within user-defined formats, 242
- `\` (backslash)
 - integer division operator, 56
 - used with escape characters, 358
 - used within filenames, 123
 - used within user-defined formats, 242
- `^` (caret), exponentiation operator, 57
- `_` (underscore), line-continuation character, 57–58
- `__stdcall` calling convention, 145
- 0** (digit), used within user-defined formats, 240
- Abs** (function), 58–59
- absolute value, 58–59
- accelerator keys, in Dialog Editor, 661
 - assigning to dialog controls, 669–71
 - testing, 679
- actions, dialog, 172, 691
- ActivateControl** (statement), 59–60
- activating
 - applications, 64–65
 - windows, 482–83
- aliases
 - used with external subroutines and functions, 146
- alignment, considerations for cross-platform scripting, 122
- And** (operator), 60–61
- annuities
 - future values of, 250
 - interest rates of, 396–97
 - number of periods for, 340–41
 - payments for, 367–68
 - present value of, 341–42, 384–85
 - principal payments for, 369–71
- AnswerBox** (function), 61–63
- antilogarithm function (**Exp**), 224
- Any** (data type), 63–64, 147
- AppActivate** (statement), 64–65
- AppClose** (statement), 65–66
- Append** (keyword), 350–53
- AppFilename\$** (function), 66
- AppFind\$** (function), 66–67
- AppGetActive\$** (function), 67
- AppGetPosition** (statement), 67–68
- AppGetState** (function), 68–69
- AppHide** (statement), 69–70
- AppleScript, executing, 312
- applications
 - activating, 64–65
 - changing size of, 75–76
 - closing, 65–66
 - finding, 66–67
 - finding active, 67
 - getting position of, 67–68
 - getting state of, 68–69
 - getting type of, 76–77
 - hiding, 69–70
 - listing, 70–71
 - maximizing, 71
 - minimizing, 71–72
 - moving, 72–73
 - restoring, 73–74
 - retrieving filenames of, 66
 - running, 426–27
 - selecting menu commands from, 315–16
 - setting state of, 74
 - showing, 75
- AppList** (statement), 70–71
- AppMaximize** (statement), 71
- AppMinimize** (statement), 71–72
- AppMove** (statement), 72–73
- AppRestore** (statement), 73–74
- AppSetState** (statement), 74
- AppShow** (statement), 75
- AppSize** (statement), 75–76
- AppType** (function), 76–77
- arctangent function (**Atn**), 84
- arguments
 - parentheses use, 49–50
 - passed to functions, 248
 - passed to subroutines, 450
 - to external routines, 93, 147, 151
- arithmetic operators. *See* operators
- arranging
 - icons, 155
 - windows
 - cascading, 155–56
 - tiling, 158–59
- ArrayDims** (function), 77–78

-
- arrays, 3
 - ArrayDims** (function), 77–78
 - declaring, 79
 - as local, 161–63
 - as private, 376–77
 - as public, 377–80
 - Dim** (statement), 161–63
 - dimensions
 - getting bounds of, 80
 - getting lower bound, 292–93
 - getting number of, 77–78, 80
 - getting upper bound, 466–67
 - LBound** (function), 292–93
 - maximum number of, 161
 - reestablishing, 398–99
 - UBound** (function), 466–67
 - dynamic, 80, 161, 376, 378, 398–99
 - erasing, 215–16
 - filling combo boxes from, 170
 - filling drop list boxes from, 170
 - filling list boxes from, 170
 - filling with application names, 70–71
 - filling with disk names, 166
 - filling with query results, 441
 - filling with window objects, 485
 - fixed-sized, declaring, 79
 - list of language elements, 3
 - operations on, 80
 - passing, 80
 - Private** (statement), 376–77
 - Public** (statement), 377–80
 - selecting items of, 414–15
 - setting default lower bound of, 356
 - size, changing while running, 398–99
 - sorting, 81
 - total size of, 161, 376, 378
 - ArraySort** (statement), 81
 - As Any** (keyword), 147
 - Asc** (function), 81–82
 - AskBox\$** (function), 82–83
 - AskPassword\$** (function), 83–84
 - assigning, objects, 420–21
 - assignment
 - = (statement), 55–56
 - Let** (statement), 296–97
 - LSet** (statement), 310–11
 - overflow during, 55, 297
 - rounding during, 226
 - RSet** (statement), 404
 - Atn** (function), 84
 - used to calculate Pi, 363
 - attributes. *See* files, attributes of
 - automation. *See* OLE automation
 - Basic.Capability** (method), 84–85, 120
 - Basic.Eoln\$** (property), 85–86
 - Basic.FreeMemory** (property), 86
 - Basic.HomeDir\$** (property), 86
 - Basic.OS** (property), 87, 120
 - Basic.PathSeparator\$** (property), 87
 - Basic.Version\$** (property), 87–88
 - BasicScript
 - free memory of, 86
 - functions to get information from, 12
 - home directory of, 86
 - version of, 87–88
 - Beep** (statement), 88
 - Begin Dialog** (statement), 88–90
 - Binary** (keyword), 350–53
 - binary data
 - reading, 251–53
 - writing, 381–84
 - binary files
 - opening, 350–53
 - reading from, 251–53
 - writing to, 381–84
 - binary operators
 - And** (operator), 60–61
 - Eqv** (operator), 214–15
 - Imp** (operator), 272–73
 - list of, 12–13
 - Not** (operator), 338–39
 - Or** (operator), 362–63
 - Xor** (operator), 494–95
 - bitmaps, used in dialog boxes, 364, 366
 - Body** (method)
 - of WMDocument, 544
 - Body** (property)
 - of WMPPoint, 616
 - Boolean** (data type), 91
 - converting to, 96
 - range of values, 91
 - storage requirements, 91
 - Boolean constants
 - False** (constant), 227
 - True** (constant), 464
 - breakpoints, in Script Editor, 640
 - removing, 642
 - setting, 640–41
 - bugs (error trapping), 219–20, 348–50
 - built-in dialogs. *See* dialogs, built-in
 - ButtonEnabled** (function), 91–92
 - ButtonExists** (function), 92

- buttons. *See also* push buttons
 - on toolbar
 - in Dialog Editor, 653–54
 - in Script Editor, 620–21
 - by value, forcing parameters, 248, 450
- ByRef** (keyword), 92–93, 146, 148, 247, 248, 449, 450
- byte ordering, with files, 121
- byte ordering, with structures, 121
- ByVal** (keyword), 49, 93–94, 146, 147, 246, 248, 449, 450
- Call** (statement), 94–95
- calling
 - external routines, 144–53
 - other routines, 94–95
- calling conventions
 - __stdcall**, 145
 - CDecl**, 145
 - for external routines, under Macintosh, 148
 - for external routines, under Win32, 148
 - for external routines, under Windows, 148
 - Pascal**, 145
 - System**, 145
- Cancel buttons
 - adding to dialog template, 95
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176
- capabilities
 - of network, 334–36
 - of platform, 84–85
- capturing
 - active application, 157–58
 - active window, 157–58
 - entire screen, 157–58
- capturing dialog boxes from another application, in Dialog Editor, 676–77
- cascading desktop windows, 155–56
- Case Else** (statement), 413
- case sensitivity, when comparing strings, 356–57
- case statement, 412–13
- CBool** (function), 96
- CCur** (function), 96–97
- cd audio, **Mci** (function), 313–15
- CDate**, **CVDate** (functions), 97
- CDbl** (function), 98
- CDecl** (keyword), 144–53
- CDecl** calling convention, 145
- character
 - codes, 81–82
 - converting to number, 81–82
- ChDir** (statement), 98–99
- ChDrive** (statement), 99
- check boxes
 - adding to dialog template, 99–101
 - checking for existence of, 101–2
 - checking if enabled, 101
 - getting state of, 178, 255
 - in Dialog Editor, 657
 - setting state of, 179, 422–23
- CheckBox** (statement), 99–101
- CheckBoxEnabled** (function), 101
- CheckBoxExists** (function), 101–2
- Choose** (function), 102–3
- Chr**, **Chr\$** (functions), 103
- chunking. *See* parsing
- CInt** (function), 103–4
- Clipboard
 - erasing, 105–6
 - getting contents of, 104–5, 106–7
 - getting type of data in, 106
 - list of language elements, 3
 - placing snapshots into, 157–58
 - setting contents of, 105, 107
- Clipboard\$** (function), 104–5
- Clipboard\$** (statement), 105
- Clipboard.Clear** (method), 105–6
- Clipboard.GetFormat** (method), 106
- Clipboard.GetText** (method), 106–7
- Clipboard.SetText** (method), 107
- CLng** (function), 107–8
- Close** (statement), 108–9
- closing
 - all files, 400
 - applications, 65–66
 - files, 108–9
 - windows, 483–84
- code resources
 - search rules for, 152
 - specifying, on the Macintosh, 153
- Collection** (topic), 497
- collections
 - defined, 346
 - elements, identifying, 346
 - indexing, 346
 - methods of, 346
 - properties of, 346
- colors, changing desktop, 156
- combo boxes
 - adding to dialog template, 109–10
 - checking for existence of, 111–12
 - checking if enabled, 110–11
 - getting edit field of, 177

- getting number of items in, 256–57
- getting selection of, 255–56
- in Dialog Editor, 657
- selecting item from, 416
- setting edit field of, 176
- setting items in, 170
- ComboBox** (statement), 109–10
- ComboBoxEnabled** (function), 110–11
- ComboBoxExists** (function), 111–12
- command line, retrieving, 112
- Command, Command\$** (functions), 112
- comments, 113
 - ' (apostrophe), 48–50
 - adding to scripts, in Script Editor, 632
 - list of language elements, 3
 - Rem** (statement), 400
- common dialogs
 - file open, 353–55
 - file save, 406–7
- comparing strings, 445–46
- comparison operators, 3–4
 - list of, 3–4
 - table of, 113
 - used with mixed types, 114
 - used with numbers, 114
 - used with strings, 114
 - used with variants, 115
- compatibility mode, opening files in, 352
- compiler errors, 705–9
- concatenation operator (&), 48
- conditionals
 - Choose** (function), 102–3
 - If...Then...Else** (statement), 270–71
 - IIf** (function), 271–72
 - Switch** (function), 451–52
- conjunction operator (**And**), 60–61
- Const** (statement), 115–17
- constants. *See also* literals
 - declaring, 115–17
 - ebAbort** (constant), 189
 - ebBold** (constant), 190
 - ebBoldItalic** (constant), 191
 - ebBoolean** (constant), 191
 - ebCurrency** (constant), 192
 - ebDate** (constant), 193
 - ebDirectory** (constant), 194–95
 - ebDOS16** (constant), 87
 - ebDouble** (constant), 195
 - ebEmpty** (constant), 195
 - ebExclamation** (constant), 195–96
 - ebHidden** (constant), 196
 - ebIgnore** (constant), 196–97
 - ebInformation** (constant), 197
 - ebInteger** (constant), 197–98
 - ebItalic** (constant), 198
 - ebLandscape** (constant), 198
 - ebLeftButton** (constant), 198–99
 - ebLong** (constant), 199
 - ebMacintosh** (constant), 87, **Error! Not a valid bookmark in entry on page 199**
 - ebMaximized** (constant), 199
 - ebMinimized** (constant), 200
 - ebNo** (constant), 200
 - ebNone** (constant), 200–201
 - ebNormal** (constant), 201
 - ebNull** (constant), 202
 - ebObject** (constant), 202
 - ebOK** (constant), 202
 - ebOKCancel** (constant), 202–3
 - ebOKOnly** (constant), 203
 - ebOS2** (constant), 87
 - ebPortrait** (constant), 203
 - ebQuestion** (constant), 203–4
 - ebReadOnly** (constant), 204
 - ebRegular** (constant), 204–5
 - ebRestored** (constant), 205
 - ebRetry** (constant), 205
 - ebRetryCancel** (constant), 206
 - ebRightButton** (constant), 206
 - ebSingle** (constant), 206–7
 - ebSolaris** (constant), 87
 - ebString** (constant), 207
 - ebSystem** (constant), 207–8
 - ebSystemModal** (constant), 208
 - ebVariant** (constant), 208
 - ebVolume** (constant), 208–9
 - ebWin16** (constant), 87, 195
 - ebWin32** (constant), 87
 - ebWindows** (constant), 209–10
 - ebYes** (constant), 210
 - ebYesNo** (constant), 210
 - ebYesNoCancel** (constant), 210–11
 - Empty** (constant), 212
 - False** (constant), 227
 - folding, 306
 - giving explicit type to, 116
 - list of, 118
 - list of language elements, 17
 - naming conventions of, 116
 - Nothing** (constant), 339
 - Null** (constant), 342–43
 - Pi** (constant), 363

- scoping of, 117
- True** (constant), 464
- control IDs, retrieving, 167
- control structures, 213
 - Do...Loop** (statement), 182–83
 - Exit Do** (statement), 221–22
 - Exit For** (statement), 222–23
 - Exit Function** (statement), 223
 - Exit Sub** (statement), 224
 - For...Next** (statement), 237–38
 - Function...End Function** (statement), 245–49
 - GoSub** (statement), 262–63, 401–2
 - Goto** (statement), 263–64
 - If...Then...Else** (statement), 270–71
 - list of, 5–6
 - Select...Case** (statement), 412–13
 - Sub...End Sub** (statement), 448–51
 - While...Wend** (statement), 481–82
- control.ini file, 156
- controlling applications
 - list of language elements, 4–5
 - Menu** (statement), 315–16
 - QueEmpty** (statement), 385–86
 - QueFlush** (statement), 386
 - QueKeyDn** (statement), 386–87
 - QueKeys** (statement), 387–88
 - QueKeyUp** (statement), 388
 - QueMouseClicked** (statement), 388–89
 - QueMouseDown** (statement), 389–90
 - QueMouseDbIClk** (statement), 390–91
 - QueMouseDbIDn** (statement), 391–92
 - QueMouseMove** (statement), 392
 - QueMouseMoveBatch** (statement), 393–94
 - QueMouseUp** (statement), 394
 - QueSetRelativeWindow** (statement), 394–95
- controlling program flow. *See* control structures
- controls. *See* dialog controls
- conversations. *See* DDE
- conversion. *See* data conversion
- coordinate systems
 - dialog base units, 407–8
 - pixels, 408–9–10
 - twips per pixel, 409
- copying
 - data
 - using = (statement), 55–56
 - using **Let** (statement), 296–97
 - using **LSet** (statement), 310–11
 - using **RSet** (statement), 404
 - files, 228–29
 - text. *See* Clipboard
 - in Script Editor, 631
 - user-defined types, 310
- Cos** (function), 118
- cosine, 118
- counters, used with **For...Next** (statement), 238
- counting
 - items in combo box, 256–57
 - items in list box, 259–60
 - items in string, 289–90
 - lines in string, 300–301
 - words, 491–92
- CreateObject** (function), 118–20
- creating dialog boxes, in Dialog Editor, 656–61
- creating new objects, 162, 337–38
- cross-platform scripting, 120–24
 - alignment, 122
 - byte ordering with files, 121
 - byte ordering with structures, 121
 - determining capabilities of platform, 120
 - determining platform, 84–85, 120
 - end-of-line character, 122
 - getting end-of-line character, 85–86
 - getting path separator, 87
 - getting platform, 87
 - path separators, 123
 - portability of compiled code, 122
 - portability of drive letters, 124
 - relative paths, 124
 - unsupported language elements, 123
- CSng** (function), 124–25
- CStr** (function), 125
- CurDir, CurDir\$** (functions), 125–26
- Currency** (data type), 126
 - converting to, 96–97
 - range of values, 126
 - storage requirements, 126
- currency format, 239
- custom controls, activating, 59–60
- custom dialogs. *See* user dialogs
- cutting text, in Script Editor, 631
- CVar** (function), 126–27
- CVDate** (function), 97
- CVErr** (function), 127–28
- data conversion
 - character to number, 81–82
 - during expression evaluation, 225
 - list of language elements, 7
 - number to character, 103
 - number to hex string, 266
 - number to octal string, 347
 - string to number, 471–72

- testing for numbers, 287–88
- to **Boolean**, 96
- to **Currency**, 96–97
- to **Date**, 97, 136, 284–85, 463–64
- to **Double**, 98
- to error, 127–28
- to **Integer**, 103–4
- to **Long**, 107–8
- to **Single**, 124–25
- to **String**, 125, 445
- to **Variant**, 126–27
- data conversion functions
 - Asc** (function), 81–82
 - CBool** (function), 96
 - CCur** (function), 96–97
 - CDate**, **CVDate** (functions), 97
 - CDBl** (function), 98
 - Chr**, **Chr\$** (functions), 103
 - CInt** (function), 103–4
 - CLng** (function), 107–8
 - CSng** (function), 124–25
 - CStr** (function), 125
 - CVar** (function), 126–27
 - CVer** (function), 127–28
 - Hex**, **Hex\$** (functions), 266
 - Oct**, **Oct\$** (functions), 347
 - Str**, **Str\$** (functions), 445
 - Val** (function), 471–72
- data objects. *See* objects
- data sources
 - retrieving DBMS of, 436
 - retrieving list of, 435
 - retrieving name of, 436
 - retrieving owner qualifier of, 436
 - retrieving server of, 436
- data types
 - Any** (data type), 63–64, 147
 - arguments. *See* arguments
 - arrays. *See* arrays
 - Boolean** (data type), 91
 - changing default, 153–55
 - converting. *See* data conversion
 - converting between. *See* data conversion
 - Currency** (data type), 126
 - Date** (data type), 129
 - Dim** (statement), 161–63
 - Double** (data type), 185–86
 - Integer** (data type), 280
 - list of, 7–8
 - Long** (data type), 309–10
 - Object** (data type), 343–44
 - Private** (statement), 376–77
 - Public** (statement), 377–80
 - returned from external functions, 146
 - Single** (data type), 427–28
 - String** (data type), 446–48
 - user-defined, 470–71
 - Variant** (data type), 472–76
- data, sharing. *See* DDE
- database
 - list of language elements, 8
- database functions
 - SQLBind** (function), 430
 - SQLClose** (function), 431
 - SQLError** (function), 432
 - SQLExecQuery** (function), 433
 - SQLGetSchema** (function), 435
 - SQLOpen** (function), 438
 - SQLRequest** (function), 439
 - SQLRetrieve** (function), 441
 - SQLRetrieveToFile** (function), 442
- databases
 - closing, 431
 - opening, 438
 - placing data, 430
 - querying, 433, 439, 441, 442
 - retrieving errors from, 432
 - retrieving information about, 435
 - retrieving list of, 435
 - retrieving list of owners of, 435
 - retrieving name of, 436
 - retrieving qualifier of, 436
 - tables
 - retrieving list of, 436
- Date** (data type), 129
 - converting to, 97, 136, 463–64
 - range of values, 129
 - specifying date constants, 129
 - storage requirements, 129
- Date**, **Date\$** (functions), 130
- Date**, **Date\$** (statements), 130–31
- date/time functions
 - Date**, **Date\$** (functions), 130
 - Date**, **Date\$** (statements), 130–31
 - DateAdd** (function), 131–33
 - DateDiff** (function), 133–34
 - DatePart** (function), 134–35
 - DateSerial** (function), 136
 - Day** (function), 136–37
 - FileDateTime** (function), 229
 - Hour** (function), 267
 - IsDate** (function), 284–85

- list of language elements, 8–9
- Minute** (function), 319–20
- Month** (function), 322–23
- Now** (function), 339–40
- Second** (function), 410
- Time, Time\$** (functions), 461
- Time, Time\$** (statements), 461–62
- Timer** (function), 462
- TimeSerial** (function), 463
- Weekday** (function), 480–81
- Year** (function), 495
- DateAdd** (function), 131–33
- DateDiff** (function), 133–34
- DatePart** (function), 134–35
- dates
 - adding, 131–33
 - converting to, 136, 284–85
 - current, 130, 339–40
 - Date** (data type), 129
 - day of month, 136–37
 - day of week, 480–81
 - file creation, 229
 - file modification, 229
 - month of year, 322–23
 - parts of, 134–35
 - reading from sequential files, 275
 - setting, 130–31
 - subtracting, 133–34
 - year, 495
- DateSerial** (function), 136
- DateValue** (function), 136
- Day** (function), 136–37
- DDB** (function), 137–38
- DDE
 - AppActivate** (statement), 64–65
 - changing timeout, 144
 - DoEvents** (function), 184
 - DoEvents** (statement), 184
 - ending conversation, 142–43
 - executing remote command, 138–39
 - getting text, 140–41
 - getting value from another application, 140–41
 - initiating conversation, 139
 - list of language elements, 9
 - sending text, 140
 - setting data in another application, 141–42
 - setting value in another application, 140
 - Shell** (function), 426–27
 - starting conversation, 139
 - terminating conversation, 142–43–44
- DDEExecute** (statement), 138–39
- DDEInitiate** (function), 139
- DDEPoke** (statement), 140
- DDERequest, DDERequest\$** (functions), 140–41
- DDESend** (statement), 141–42
- DDETerminate** (statement), 142–43
- DDETerminateAll** (statement), 143–44
- DDETimeout** (statement), 144
- debugger, invoking, 444
- debugging scripts, in Script Editor, 638. *See also* Script Editor
 - breakpoints, 640
 - removing, 642
 - setting, 640–41
 - instruction pointer, 638
 - moving to another line in subroutine, 639
 - procedure calls, tracing, 639
 - script execution, tracing, 638–39
 - watch variables, 642
 - adding, 642–44
 - deleting, 644
 - modifying value of, 644–45
 - selecting, 644
- decision making. *See also* control structures
 - Choose** (function), 102–3
 - If...Then...Else** (statement), 270–71
 - IIf** (function), 271–72
 - Select...Case** (statement), 412–13
 - Switch** (function), 451–52
- Declare** (statement), 63, 144–53
- declaring
 - implicit variables, 161
 - object variables, 162, 337–38, 343, 345
 - with **Dim** (statement), 161–63
 - with **Private** (statement), 376–77
 - with **Public** (statement), 377–80
- default data type, changing, 153–55
- default properties, 226
- DefType** (statement), 153–55
- degrees, converting to radians, 84
- DELETE** (SQL statement), 434, 440
- deleting controls, in Dialog Editor, 674
- deleting text, in Script Editor, 630–31
- delimited files, reading, 274–76
- depreciation
 - calculated using double-declining balance method, 137–38
 - straight-line depreciation, 429
 - sum of years' digits depreciation, 452–53
- Desktop.ArrangeIcons** (method), 155
- Desktop.Cascade** (method), 155–56
- Desktop.SetColors** (method), 156

-
- Desktop.SetWallpaper** (method), 156–57
 - Desktop.Snapshot** (method), 157–58
 - Desktop.Tile** (method), 158–59
 - Dialog** (function), 159–60
 - Dialog** (statement), 160–61
 - dialog actions, 172, 691
 - dialog boxes, in Dialog Editor. *See also* dialog controls; dialog controls, in Dialog Editor; Dialog Editor; user dialogs
 - attributes of, adjusting with Information dialog box, 664–65
 - capturing from another application, 676–77
 - creating, 656–61
 - editing, 651, 661–74
 - incorporating dialog boxes or dialog controls into script, 679
 - Information dialog box for, displaying, 662–63
 - moving
 - with arrow keys, 667
 - with Information dialog box, 667
 - with mouse, 666–67
 - opening dialog box template files in Dialog Editor, 677
 - pasting dialog box controls into Dialog Editor, 675–76
 - pasting dialog boxes into Dialog Editor, 675, 676
 - resizing, 668
 - with Information dialog box, 668
 - with mouse, 668
 - selecting, 662
 - testing, 678
 - for basic problems, 677–78
 - for operational problems, 678–79
 - when there are hidden controls, 668
 - titles of, changing, 669
 - dialog callback. *See* dialog procedures
 - dialog controls. *See also* dialog controls, in Dialog Editor
 - activating, 59–60
 - adding, in Dialog Editor, 658–59
 - Cancel buttons
 - adding to dialog template, 95
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176
 - changing focus of, 169–70
 - changing text of, 176–77
 - check boxes
 - adding to dialog template, 99–101
 - checking existence of, 101–2
 - checking if enabled, 101
 - getting state of, 178, 255
 - in Dialog Editor, 657
 - setting state of, 179, 422–23
 - combo boxes
 - adding to dialog template, 109–10
 - checking for existence of, 111–12
 - checking if enabled, 110–11
 - getting edit field of, 177
 - getting number of items in, 256–57
 - getting selection of, 255–56
 - in Dialog Editor, 657
 - selecting item from, 416
 - setting edit field of, 176
 - setting items in, 170
 - deleting, in Dialog Editor, 674
 - disabling, 168–69
 - drop list boxes
 - adding to dialog template, 186–88
 - getting selection index of, 178
 - getting selection of, 177
 - in Dialog Editor, 657
 - setting items in, 170
 - setting selection of, 176, 179
 - duplicating, in Dialog Editor, 673
 - enabling, 168–69
 - getting enabled state of, 167–68
 - getting focus of, 169
 - getting text of, 177–78
 - getting value of, 178
 - getting visibility of, 179–80
 - group boxes
 - adding to dialog template, 265–66
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176
 - list boxes
 - adding to dialog template, 301–3
 - checking for existence of, 303–4
 - checking if enabled, 303
 - getting number of items in, 259–60
 - getting selection index of, 178
 - getting selection of, 177, 258–59
 - in Dialog Editor, 657
 - selecting item from, 416–17
 - setting items in, 170
 - setting selection of, 176, 179
 - list of language elements, 9–10
 - moving, in Dialog Editor, 666–67–68
 - OK buttons
 - adding to dialog template, 347–48
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176

- option buttons
 - adding to dialog template, 358–59
 - checking existence of, 360–61
 - checking if enabled, 359–60
 - getting label of, 177
 - getting selection index of, 178
 - getting state of, 261–62
 - grouping within dialog template, 361–62
 - in Dialog Editor, 657
 - selecting, 179
 - setting label of, 176
 - setting state of, 424–25
- picture button controls
 - adding to dialog template, 365–67
- picture buttons
 - in Dialog Editor, 657
- picture controls
 - adding to dialog template, 363–65
 - in Dialog Editor, 657
 - setting image of, 174–76
- positioning with grid, in Dialog Editor, 659–60
- push buttons
 - adding to dialog template, 380–81
 - checking for existence of, 92
 - checking if enabled, 91–92
 - getting label of, 177
 - in Dialog Editor, 657
 - selecting, 415
 - setting label of, 176
- resizing, in Dialog Editor, 668–69
- retrieving ID of, 167
- selecting, in Dialog Editor, 662
- setting value of, 178–79
- setting visibility of, 180–82
- text boxes
 - adding to dialog template, 459–61
 - getting content of, 177, 257–58
 - in Dialog Editor, 657, 679
 - setting content of, 176, 423–24
- text controls
 - adding to dialog template, 458–59
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176
- dialog controls, in Dialog Editor. *See also* dialog boxes, in Dialog Editor; dialog controls; Dialog Editor; user dialogs
 - accelerator keys
 - assigning to, 669–71
 - testing, 679
 - adding to dialog box, 658–59
 - attributes of, adjusting with Information dialog box, 664, 665–66
 - Cancel buttons, 657
 - check boxes, 657
 - combo boxes, 657
 - creating efficiently, 660–61
 - deleting
 - all controls, 674
 - one control, 674
 - drop list boxes, 657
 - duplicating, 673
 - group boxes, 657
 - hidden, 668
 - Information dialog box for, displaying, 663–64
 - labels of, changing, 669
 - list boxes, 657
 - moving
 - with arrow keys, 667
 - with Information dialog box, 667–68
 - with mouse, 666–67
 - OK buttons, 657
 - option buttons, 657
 - grouping, 661, 679
 - pasting controls into Dialog Editor, 675–76
 - picture buttons, 657. *See also* dialog controls, in Dialog Editor, specifying pictures
 - picture controls, 657. *See also* dialog controls, in Dialog Editor, specifying pictures
 - positioning with grid, 659–60
 - push buttons, 657
 - resizing, 668
 - automatically, 668–69
 - with Information dialog box, 668
 - with mouse, 668
 - selecting, 662
 - specifying pictures, 671
 - from file, 671
 - from picture library, 671–72
 - tabbing order of, 660, 676, 678
 - text boxes, 657, 679
 - text controls, 657
 - types of, 656–57
- Dialog Editor, 651, 652. *See also* dialog boxes, in Dialog Editor; dialog controls; dialog controls, in Dialog Editor; user dialogs
 - application window, 652–53
 - Controls menu, 683–84
 - controls supported by, 656–57
 - Dialog Translation Errors dialog box, 676, 677
 - Edit menu, 682–83
 - exiting from, 680

- features of, 652
- File menu, 680–81
- grid, displaying and adjusting, 659–60
- help
 - for current window, 655
 - on selected topics, 655–56
- Help menu, 684–85
- Information dialog box, 662–66
 - adjusting control attributes with, 664, 665–66
 - adjusting dialog box attributes with, 664–65
 - displaying
 - for controls, 663–64
 - for dialog boxes, 662–63
- keyboard shortcuts, 654–55
- Pick tool, 685
- picture libraries, 672–73
 - creating for use in, 672–73
 - modifying for use in, 673
- toolbar of, 653–54
- undoing editing operations, 674
- dialog procedures, 171–74
 - actions sent to, 172, 691
- dialog templates. *See* user dialogs
- dialog units, calculating, 407–8
- dialogs, built-in. *See also* user dialogs
 - AnswerBox** (function), 61–63
 - AskBox\$** (function), 82–83
 - AskPassword\$** (function), 83–84
 - InputBox**, **InputBox\$** (functions), 277–78
 - listing of, 14–15
 - MsgBox** (function), 323–26
 - MsgBox** (statement), 327
 - MsgClose** (statement), 327
 - MsgOpen** (statement), 327–29
 - MsgSetText** (statement), 329
 - MsgSetThermometer** (statement), 329–30
 - PopupMenu** (function), 368–69
 - SaveFilename\$** (function), 406–7
 - SelectBox** (function), 414–15
 - user-defined, 88–90
- Dim** (statement), 161–63
- dimensions. *See* arrays, dimensions
- Dir**, **Dir\$** (functions), 163–65
- directories
 - changing, 98–99
 - containing BasicScript, 86
 - containing Windows, 455
 - creating, 321–22
 - getting list of, 230
 - getting path separator, 87
 - parsing names of, 234–35
 - removing, 403
 - retrieving, 125–26
 - retrieving filenames from, 163–65, 231–34
- disabling, dialog controls, 168–69
- disjunction operator (**Or**), 362–63
- disk drives
 - changing, 99
 - getting free space of, 166–67
 - platform support, 124
 - retrieving current directory of, 125–26
 - retrieving list of, 166
- DiskDrives** (statement), 166
- DiskFree** (function), 166–67
- displaying messages, 323–26, 327
 - breaking text across lines, 325
- DlgControlId** (function), 167
- DlgEnable** (function), 167–68
- DlgEnable** (statement), 168–69
- DlgFocus** (function), 169
- DlgFocus** (statement), 169–70
- DlgListBoxArray** (function), 170
- DlgProc** (function), 171–74
- DlgSetPicture** (statement), 174–76
- DlgText** (statement), 176–77
- DlgText\$** (function), 177–78
- DlgValue** (function), 178
- DlgValue** (statement), 178–79
- DlgVisible** (function), 179–80
- DlgVisible** (statement), 180–82
- DLLs. *See also* external routines
 - calling, 144–53
 - Declare** (statement), 144–53
 - search rules for, under Windows, 152
- Do...Loop** (statement), 182–83
 - exiting **Do** loop, 221–22
- documentation. *See* comments
- DoEvents** (function), 184
- DoEvents** (statement), 184
- Double** (data type), 185–86
 - converting to, 98
 - internal format, 186
 - range of values, 186
 - storage requirements, 185–86
- double-declining balance method, used to calculate depreciation, 137–38
- drives. *See* disk drives
- drop list boxes
 - adding to dialog template, 186–88
 - getting selection index of, 178
 - getting selection of, 177
 - in Dialog Editor, 657

- setting items in, 170
- setting selection of, 176, 179
- DropListBox** (statement), 186–88
- duplicating controls, in Dialog Editor, 673
- dynamic arrays, 80
- dynamic data exchange. *See* DDE
- dynamic dialogs. *See* user dialogs
- dynamic link libraries. *See* DLLs
- e. *See* logarithms
- ebAbort** (constant), 189
- ebBold** (constant), 190
- ebBoldItalic** (constant), 191
- ebBoolean** (constant), 191
- ebCurrency** (constant), 192
- ebDate** (constant), 193
- ebDirectory** (constant), 194–95
- ebDOS** (constant), 87
- ebDouble** (constant), 195
- ebEmpty** (constant), 195
- ebExclamation** (constant), 195–96
- ebHidden** (constant), 196
- ebIgnore** (constant), 196–97
- ebInformation** (constant), 197
- ebInteger** (constant), 197–98
- ebItalic** (constant), 198
- ebLandscape** (constant), 198
- ebLeftButton** (constant), 198–99
- ebLong** (constant), 199
- ebMacintosh** (constant), 87, **Error! Not a valid bookmark in entry on page 199**
- ebMaximized** (constant), 199
- ebMinimized** (constant), 200
- ebNo** (constant), 200
- ebNone** (constant), 200–201
- ebNormal** (constant), 201
- ebNull** (constant), 202
- ebObject** (constant), 202
- ebOK** (constant), 202
- ebOKCancel** (constant), 202–3
- ebOKOnly** (constant), 203
- ebOS2** (constant), 87
- ebPortrait** (constant), 203
- ebQuestion** (constant), 203–4
- ebReadOnly** (constant), 204
- ebRegular** (constant), 204–5
- ebRestored** (constant), 205
- ebRetry** (constant), 205
- ebRetryCancel** (constant), 206
- ebRightButton** (constant), 206
- ebSingle** (constant), 206–7
- ebSolaris** (constant), 87
- ebString** (constant), 207
- ebSystem** (constant), 207–8
- ebSystemModal** (constant), 208
- ebVariant** (constant), 208
- ebVolume** (constant), 208–9
- ebWin16** (constant), 87, 195
- ebWin32** (constant), 87
- ebWindows** (constant), 209–10
- ebYes** (constant), 210
- ebYesNo** (constant), 210
- ebYesNoCancel** (constant), 210–11
- edit controls. *See* text boxes
- EditEnabled** (function), 211
- EditExists** (function), 211–12
- editing custom dialog boxes, 651, 661–74
- Else** (keyword), 270–71
- ElseIf** (keyword), 270–71
- embedded objects. *See* OLE automation
- embedded quotation marks, 305
- Empty** (constant), 212
- Empty**, testing for, 285
- enabling, dialog controls, 168–69
- End** (statement), 213
- end of file
 - checking, 213–14
 - checking for, 213–14
- end-of-line, in sequential files, 276
- entry points, **Main** (statement), 313
- Environ**, **Environ\$** (functions), 213
- environment variables, getting, 213
- environment, controlling
 - list of language elements, 6–7
- EOF** (function), 213–14
- equivalence operator (**Eqv**), 214–15
- Eqv** (operator), 214–15
- Erase** (statement), 215–16
- Erl** (function), 216–17
- Err** (function), 217
- Err** (statement), 218
- Error** (statement), 218–19
- error handlers
 - cascading, 220
 - nesting, 219, 349
 - removing, 349
 - resetting, 218, 349
 - resuming, 349, 400–401
- error messages
 - BasicScript-specific, 702, 703
 - compatible with Visual Basic, 699
 - compiler, 705–9
 - runtime, 699–702

-
- error trapping, 219–20, 348–50
 - Error, Error\$** (functions), 220–21
 - errors
 - BasicScript-specific, 220
 - cascading, 220
 - Erl** (function), 216–17
 - Err** (function), 217
 - Err** (statement), 218
 - Error** (statement), 218–19
 - Error, Error\$** (functions), 220–21
 - generating, 218–19
 - getting error number of, 217
 - getting line number of, 216–17
 - getting text of, 220–21
 - handling, 219–20
 - list of language elements, 10
 - On Error** (statement), 348–50
 - range of values for, 218
 - resetting state of, 218
 - Resume** (statement), 400–401
 - resuming control after, 220
 - setting, 218
 - SQL, 432
 - Stop** (statement), 444
 - trapping, 348–50
 - user-defined, 220
 - converting to, 127–28
 - printing, 371
 - printing to sequential files, 373
 - reading from binary/random files, 252
 - reading from sequential files, 275
 - testing for, 285–86
 - writing to random/binary files, 383
 - writing to sequential files, 492
 - Visual Basic compatibility with, 220
 - escape characters, table of, 358
 - exclusive or operator (**Xor**), 494–95
 - executing scripts, in Script Editor
 - pausing execution of, 637
 - starting execution of, 637
 - stopping execution of, 637
 - tracing execution of, 638–39
 - Exit Do** (statement), 182, 221–22
 - Exit For** (statement), 222–23, 238
 - Exit Function** (statement), 223
 - Exit Sub** (statement), 224
 - exiting
 - from Dialog Editor, 680
 - from Script Editor, 646
 - exiting from Dialog Editor, 680
 - exiting operating environment, 453
 - Exp** (function), 224
 - exponentiation operator (^), 57
 - expressions
 - evaluation of, 225–26
 - promotion of operands within, 225
 - propagation of **Null** through, 342
 - external routines
 - calling, 144–53
 - calling conventions of, 148
 - passing parameters, 147
 - data formats, 149
 - null pointers, 149
 - strings, 148
 - using **ByVal** (keyword), 93, 151
 - specified with ordinal numbers, 152
 - under Macintosh, 152
 - under Windows, 152
 - False** (constant), 227
 - file I/O
 - Close** (statement), 108–9
 - EOF** (function), 213–14
 - Get** (statement), 251–53
 - Input#** (statement), 274–76
 - Line Input#** (statement), 298–99
 - Loc** (function), 306–7
 - Lock** (statement), 307–8
 - Lof** (function), 308–9
 - Open** (statement), 350–53
 - Print#** (statement), 371–72
 - Put** (statement), 381–84
 - Reset** (statement), 400
 - Seek** (function), 410–11
 - Seek** (statement), 411–12
 - Spc** (function), 430
 - Tab** (function), 457
 - Unlock** (statement), 467–69
 - Width#** (statement), 482
 - Write#** (statement), 492–93
 - file numbers, finding available, 245
 - file open dialog box, 353–55
 - file save dialog box, 406–7
 - file system
 - list of language elements, 11–12
 - FileAttr** (function), 227–28
 - FileCopy** (statement), 228–29
 - FileDateTime** (function), 229
 - FileDirs** (statement), 230
 - FileExists** (function), 230–31
 - FileLen** (function), 231
 - FileList** (statement), 231–34
 - FileParse\$** (function), 234–35

files

attributes of

- ebArchive** (constant), 190
- ebDirectory** (constant), 194–95
- ebHidden** (constant), 196
- ebNone** (constant), 200–201
- ebNormal** (constant), 201
- ebReadOnly** (constant), 204
- ebSystem** (constant), 207–8
- ebVolume** (constant), 208–9
- getting, 253–55
- setting, 421–22
- used with **Dir**, **Dir\$** (functions), 165
- used with **FileList** (statement), 233
- used with **GetAttr** (function), 254

attributes, used with **SetAttr** (statement), 422

checking existence of, 230–31

checking for end of, 213–14

closing, 108–9

closing all, 400

copying, 228–29

deleting, 291–92

end-of-line character, 122

getting date and time of, 229

getting length of, 231

getting list of, 163–65, 231–34

getting mode of, 227–28

getting next available file number, 245

getting position within, 306–7, 410–11

getting size of, 308–9

list of language elements, 10–11

locking regions in, 307–8

opening, 350–53

- access capabilities, 352

- modes, 351

- setting another process's access rights, 352

- setting record length, 352

- truncating to zero length, 351

printing, 375–76

reading, 274–76

reading binary data from, 251–53

reading lines from, 298–99

renaming, 330–31

requesting name of, 406–7

setting read/write position in, 411–12

sharing, 352

splitting names of, 234–35

types of

- ebWindows** (constant), 209–10

- FileType** (function), 236

- getting, 236

unlocking regions in, 467–69

writing binary data to, 381–84

writing query results to, 442

writing to, 371–72, 492–93

FileType (function), 236

financial functions

- DDB** (function), 137–38

- Fv** (function), 250

- IPmt** (function), 280–82

- IRR** (function), 282–83

- list of, 12

- MIRR** (function), 320–21

- NPer** (function), 340–41

- Npv** (function), 341–42

- Pmt** (function), 367–68

- PPmt** (function), 369–71

- Pv** (function), 384–85

- Rate** (function), 396–97

- Sln** (function), 429

- SYD** (function), 452–53

finding

- applications, 66–67

- files, 163–65

- strings, 278–79

- windows, 484–85

Fix (function), 236–37. *See also* **Int** (function)

fixed arrays, 79

fixed numeric format, 239

fixed-length strings

- conversion between variable-length, 447

- declaring, 161, 376, 378

- passing to external routines, 148, 150

- within structures, 465

floating-point values

- Double** (data type), 185–86

- Single** (data type), 427–28

focus, of dialog controls

- getting, 169

- setting, 169–70

fonts, within user-dialogs, 90

For...Next (statement), 237–38

- exiting **For** loop, 222–23

formatting data

- built-in, 239

- built-in formats

 - date/time, 240

 - numeric, 239

- in files

 - Spc** (function), 430

 - Tab** (function), 457

 - Width#** (statement), 482

- user-defined formats, 240
 - date/time, 243
 - numeric, 240
 - string, 242
- forward referencing, with **Declare** (statement), 63, 144–53
- FreeFile** (function), 245
- Function...End Function** (statement), 245–49
- Function...End Sub** (statement), exiting function, 223
- Functions, 24
 - defining, 245–49
 - exiting function, 223
 - naming conventions of, 246
 - returning values from, 247
- future value of annuity, calculating, 250
- fuzzy string comparisons, 297–98
- Fv** (function), 250
- general date format, 240
- general number format, 239
- generating random numbers, 395
- Get** (statement), 251–53
- GetAttr** (function), 253–55
- GetCheckBox** (function), 255
- GetComboBoxItem\$** (function), 255–56
- GetComboBoxItemCount** (function), 256–57
- GetEditText\$** (function), 257–58
- GetListBoxItem\$** (function), 258–59
- GetListBoxItemCount** (function), 259–60
- GetOption** (function), 261–62
- global (public) variables, 377–80
- Global** (statement) (**Public** [statement]), 377–80
- GoSub** (statement), 262–63
 - returning from, 401–2
- Goto** (statement), 263–64
- grep (**Like** [operator]), 297–98
- grid, in Dialog Editor, 659–60
- group boxes
 - adding to dialog template, 265–66
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176
- GroupBox** (statement), 265–66
- grouping option buttons, 361–62
- handles, getting operating system file handles, 227–28
- height, of screen, 408–9
- help
 - in Dialog Editor, 655–56, 684–85
 - in Script Editor, 624–26, 649
- Hex**, **Hex\$** (functions), 266
- hexadecimal characters, in strings, 358
- hexadecimal strings
 - converting to, 266
 - converting to numbers, 471–72
- hiding
 - applications, 69–70
 - dialog controls, 180–82
- HLine** (statement), 266–67
- home directory, 86
- Hour** (function), 267
- HPage** (statement), 267–68
- HScroll** (statement), 268
- HWND** (object), 268–69
- HWND** (object), getting value of, 269–70
- icons, arranging on desktop, 155
- idle loops
 - DoEvents** (function), 184
 - DoEvents** (statement), 184
- If...Then...Else** (statement), 270–71
- If...Then...End If** (statement), shorthand for (**IIf**), 271–72
- IIf** (function), 271–72
- Imp** (operator), 272–73
- implication operator (**Imp**), 272–73
- implicit variable declaration, with **DefType** (statement), 153–55
- indexing collections, 346
- infinite loops, breaking out of, 183, 238, 482
- Information dialog box, in Dialog Editor, 662–66
- ini files. *See also* win.ini file; control.ini file
 - list of language elements, 12
 - reading items from, 397
 - reading section names from, 397–98
 - writing items to, 493–94
- Inline** (statement), 273–74
- Input** (keyword), 350–53
- Input#** (statement), 274–76
- InputBox**, **InputBox\$** (functions), 277–78
- INSERT** (SQL statement), 434, 440
- inserting text, in Script Editor, 628
- insertion point, moving, in Script Editor, 626–27
 - to specified line, 627–28
 - with keyboard, 622–23
 - with mouse, 627
- instantiation of OLE objects, 118–20
- InStr** (function), 278–79
- Int** (function), 279–80. *See also* **Fix** (function)
- Integer** (data type), 280
 - converting to, 103–4
 - range of values for, 280
 - storage requirements of, 280
- integer division operator (****), 56
- Interactive Operation, 20

- intercepting (trapping) errors, 219–20, 348–50
- interest payments, calculating, 280–82
- internal rate of return, calculating, 282–83, 320–21
- IPmt** (function), 280–82
- IRR** (function), 282–83
- Is** (operator), 283–84
- IsDate** (function), 284–85
- IsEmpty** (function), 285
- IsError** (function), 285–86
- IsMissing** (function), 249, 286–87, 451
- IsNull** (function), 287
- IsNumeric** (function), 287–88
- IsObject** (function), 288–89
- Item\$** (function), 289
- ItemCount** (function), 289–90
- iterating through collections, 346
- jumps
 - GoSub** (statement), 262–63
 - Goto** (statement), 263–64
 - Return** (statement), 401–2
- keyboard shortcuts
 - in Dialog Editor, 654–55
 - in Script Editor, 621–24
- keystrokes, sending
 - DoEvents** (function), 184
 - DoEvents** (statement), 184
 - QueKeyDn** (statement), 386–87
 - restrictions, 388
 - special characters, 419
 - to applications, 185, 386–87–88
- keywords
 - list of, 291
 - restrictions for, 291
- Kill** (statement), 291–92
- labels
 - in place of line numbers, 299
 - naming conventions of, 264
 - used with **GoSub** (statement), 262
 - used with **Goto** (statement), 264
- LBound** (function), 292–93
 - used with OLE arrays, 293
- LCase**, **LCase\$** (functions), 293–94
- least precise operand, 356
- Left**, **Left\$** (functions), 294
- Len** (function), 294–96
- Len** (keyword), specifying record length, 350–53
- Let** (statement), 296–97
- Lib** (keyword), 144–53
- Like** (operator), 297–98
- line breaks, in **MsgBox** (statement), 325
- line continuation, 57–58
 - in Script Editor, 633
- Line Input#** (statement), 298–99
- line numbers, 299
- Line\$** (function), 299–300
- LineCount** (function), 300–301
- linking. *See* DDE; OLE automation
- list boxes
 - adding to dialog template, 301–3
 - checking for existence of, 303–4
 - checking if enabled, 303
 - getting number of items in, 259–60
 - getting selection index of, 178
 - getting selection of, 177, 258–59
 - in Dialog Editor, 657
 - selecting item from, 416–17
 - setting items in, 170
 - setting selection of, 176, 179
- ListBox** (statement), 301–3
- ListBoxEnabled** (function), 303
- ListBoxExists** (function), 303–4
- literals, 305–6
- Loc** (function), 306–7
- local variables. *See also* variables
 - declaring, 161–63
- Lock** (statement), 307–8
- locking file regions, 307–8
- Lof** (function), 308–9
- Log** (function), 309
- logarithm function (**Log**), 309
- logarithms
 - Exp** (function), 224
 - Log** (function), 309
- logical constants
 - False** (constant), 227
 - True** (constant), 464
- logical negation, 338–39
- logical operators
 - And** (operator), 60–61
 - Eqv** (operator), 214–15
 - Imp** (operator), 272–73
 - list of, 12–13
 - Not** (operator), 338–39
 - Or** (operator), 362–63
 - Xor** (operator), 494–95
- Long** (data type), 309–10
 - converting to, 107–8
 - range of values, 310
 - storage requirements for, 310
- long date format, 240
- long time format, 240
- looping

- Do...Loop** (statement), 182–83
 - exiting **Do** loop, 221–22
 - exiting **For** loop, 222–23
- For...Next** (statement), 237–38
- lowercasing strings, 293–94
- LSet** (statement), 310–11
- LTrim**, **LTrim\$** (functions), 311
- MacID** (function), 65, 165, 292, 311–12, 427
- Macintosh, **MacID** (function), 311–12
- MacScript** (statement), 312
- Main** (statement), 313
- matching strings, 297–98
- math functions
 - Abs** (function), 58–59
 - Atn** (function), 84
 - Cos** (function), 118
 - Exp** (function), 224
 - Fix** (function), 236–37
 - Int** (function), 279–80
 - list of, 13
 - Log** (function), 309
 - Randomize** (statement), 395–96
 - Rnd** (function), 403–4
 - Sgn** (function), 425–26
 - Sin** (function), 427
 - Sqr** (function), 444
 - Tan** (function), 457–58
- math operators. *See* operators
- maximizing
 - applications, 71
 - windows, 485–86
- Mci** (function), 313–15
- medium date format, 240
- medium time format, 240
- memory
 - available, 453–54
 - available resources, 454
 - available within BasicScript, 86
 - total, 455
 - total size for arrays, 161
- Menu** (statement), 315–16
- MenuItemChecked** (function), 316
- MenuItemEnabled** (function), 317
- MenuItemExists** (function), 317
- menus
 - determining existence of, 317
 - determining if checked, 316
 - determining if enabled, 317
 - pop-up, 368–69
 - selecting, 315–16
- menus, reference for
 - in Dialog Editor, 680–85
 - in Script Editor, 646–49
- message dialog
 - changing text of, 329
 - closing, 327
 - creating, 327–29
 - setting thermometer, 329–30
- messages, error. *See* error messages
- messages, runtime error, 699–702
- metafiles
 - used in dialog boxes, 364, 366
 - used with picture controls, 175, 365, 367
- Methods, 24
 - defined, 344
 - invoking, 345
 - with OLE automation, 343
- Mid**, **Mid\$** (functions), 317–18
- Mid**, **Mid\$** (statements), 318–19
- minimizing
 - applications, 71–72
 - windows, 486–87
- Minute** (function), 319–20
- MIRR** (function), 320–21
- MkDir** (statement), 321–22
- Mod** (operator), 322
- modeless message dialog, 328
- modes, for open files, 227–28
- Month** (function), 322–23
- most precise operand, 356
- mouse
 - clicking button, 388–89
 - double-clicking button, 389–90
 - double-pressing button, 390–91
 - moving, 392
 - moving in batch, 393–94
 - pressing button, 391–92
 - releasing button, 394
 - setting coordinates relative to window, 394–95
 - trails, setting, 454
- moving
 - applications, 72–73
 - controls, in Dialog Editor, 666–67–68
 - dialog boxes, in Dialog Editor, 666–67
 - windows, 487–88
- MsgBox** (function), 323–26
- MsgBox** (statement), 327
 - constants used with
 - vbAbort** (constant), 189
 - vbArchive** (constant), 190
 - vbCancel** (constant), 191–92
 - vbCritical** (constant), 192

- ebDataObject** (constant), 192–93
- ebDefaultButton1** (constant), 194
- ebDefaultButton2** (constant), 194
- ebDefaultButton3** (constant), 194
- ebExclamation** (constant), 195–96
- ebIgnore** (constant), 196–97
- ebInformation** (constant), 197
- ebNo** (constant), 200
- ebOK** (constant), 202
- ebOKCancel** (constant), 202–3
- ebOKOnly** (constant), 203
- ebQuestion** (constant), 203–4
- ebRetry** (constant), 205
- ebRetryCancel** (constant), 206
- ebSystemModal** (constant), 208
- ebYes** (constant), 210
- ebYesNo** (constant), 210
- ebYesNoCancel** (constant), 210–11
- MsgClose** (statement), 327
- MsgOpen** (statement), 327–29
- MsgSetText** (statement), 329
- MsgSetThermometer** (statement), 329–30
- multidimensional arrays. *See* arrays
- Name** (statement), 330–31
- naming conventions
 - of constants, 116
 - of functions, 246
 - of labels, 264
 - of subroutines, 449
 - of variables, 163
- negation
 - logical, 338–39
 - unary minus operator, 52–53
- nesting, **For...Next** (statement), 238
- net present value, calculating, 341–42
- Net.AddCon** (method), 331–32
- Net.Browse\$** (method), 332–33
- Net.CancelCon** (method), 333
- Net.Dialog** (method), 333–34
- Net.GetCaps** (method), 334–36
- Net.GetCon\$** (method), 337
- Net.User\$** (property), 337
- networks
 - canceling connection, 333
 - capabilities of, 334–36
 - getting name of connection, 337
 - getting user name, 337
 - invoking browse dialog box, 332–33
 - invoking network dialog, 333–34
 - list of language elements, 13–14
 - redirecting local device, 331–32
- New** (keyword), 162, 337–38, 420–21
- Next** (keyword), 237–38
- Not** (operator), 338–39
- Nothing** (constant), 339
 - used with **Is** (operator), 283
- Now** (function), 339–40
- NPer** (function), 340–41
- Npv** (function), 341–42
- Null**
 - checking for, 287
 - propagation of, 342
 - vs. **Empty**, 342–43
- Null** (constant), 342–43
- nulls, embedded within strings, 447
- numbers
 - adding, 51
 - converting from strings, 471–72
 - converting to strings, 445
 - floating-point, 185–86, 427–28
 - getting sign of, 425–26
 - hexadecimal representation, 305
 - IsNumeric** (function), 287–88
 - octal representation, 305
 - printing, 371–72
 - reading from binary/random files, 251–53
 - reading from sequential files, 274–76
 - testing for, 287–88
 - truncating, 236–37, 279–80
 - writing to binary/random files, 381–84
 - writing to sequential files, 371–72, 492–93
- numeric operators
 - (operator), 52–53
 - \ (operator), 56
 - * (operator), 50
 - + (operator), 50–52
 - / (operator), 54–55
 - ^ (operator), 57
 - list of, 14
- Object** (data type), 343–44
 - storage requirements for, 343
- object collections. *See* collections
- objects, 22, 344–47. *See also* OLE automation
 - accessing methods of, 345
 - accessing properties of, 343, 345
 - assigning, 420–21
 - assigning values to, 345
 - automatic destruction, 344
 - collections of, 346
 - comparing, 283–84, 346
 - creating, 420–21
 - creating new, 162, 337–38

- declaring, 161–63, 343, 345, 376–77
- declaring as public, 377–80
- defined, 344
- instantiating, 343
- invoking methods of, 343
- list of language elements, 14
- OLE, creating, 118–20
- predefined, table of, 347
- testing for, 288–89
- testing if uninitialized, 283
- using dot separator, 343
- Oct, Oct\$** (functions), 347
- octal characters, in strings, 358
- octal strings
 - converting to, 347
 - converting to numbers, 471–72
- OK buttons
 - adding to dialog template, 347–48
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176
- OKButton** (statement), 347–48
- OLE automation. *See also* DDE
 - automatic destruction, 344
 - CreateObject** (function), 118–20
 - creating objects, 118–20
 - default properties of, 226
 - Object** (data type), 343–44
 - Set** (statement), 420–21
- On Error** (statement), 219, 348–50
- on/off format, 240
- Open** (statement), 350–53
- operating environment
 - exiting, 453
 - free memory of, 453–54
 - free resources of, 454
 - restarting, 454–55
 - total memory in, 455
- operators
 - (operator), 52–53
 - &** (operator), 48
 - \ (operator), 56
 - * (operator), 50
 - + (operator), 50–52
 - / (operator), 54–55
 - < (operator), 3–4
 - <= (operator), 3–4
 - <> (operator), 3–4
 - = (operator), 3–4
 - > (operator), 3–4
 - >= (operator), 3–4
 - ^ (operator), 57
 - And** (operator), 60–61
 - Eqv** (operator), 214–15
 - Imp** (operator), 272–73
 - Is** (operator), 283–84
 - Like** (operator), 297–98
 - Mod** (operator), 322
 - Not** (operator), 338–39
 - Or** (operator), 362–63
 - precedence of, 355
 - precision of, 356
 - Xor** (operator), 494–95
- Option Base** (statement), 161, 356, 376, 378
- option buttons
 - adding to dialog template, 358–59
 - checking existence of, 360–61
 - checking if enabled, 359–60
 - getting label of, 177
 - getting selection index of, 178
 - getting state of, 261–62
 - grouping within dialog template, 361–62
 - in Dialog Editor, 657, 661, 679
 - selecting, 179
 - setting label of, 176
 - setting state of, 424–25
- Option Compare** (statement), 356–57
 - effect on **InStr** (function), 279
 - effect on **Like** (operator), 297
 - effect on string comparisons, 114, 446
- Option CStrings** (statement), 358
- Optional** (keyword), 146, 246, 449
- optional parameters
 - checking for, 286–87
 - passed to functions, 248
 - passed to subroutines, 450
 - passing to external routines, 146
 - passing to functions, 246
 - passing to subroutines, 449
- OptionButton** (statement), 358–59
- OptionEnabled** (function), 359–60
- OptionExists** (function), 360–61
- OptionGroup** (statement), 361–62
- Or** (operator), 362–63
- ordinal values, 152
- Output** (keyword), 350–53
- overflow, in assignment, 55, 297
- pane, in Script Editor
 - edit, 620
 - separator, 620
 - watch, 620
- Parameters, 25

- passing by reference, 92–93
 - passing by value, 49–50, 93–94
 - to external routines, 93, 147, 151
- parentheses, used in expressions, 49–50
- parsing
- filenames, 234–35
 - list of language elements, 14
 - strings
 - by item, 289
 - by line, 299–300
 - by words, 490–91
 - counting items within, 289–90
 - counting lines within, 300–301
 - counting words within, 491–92
- Pascal** calling convention, 145
- password, requesting from user, 83–84
- pasting text. *See also* Clipboard
- in Script Editor, 631
- path separator
- getting, 87
 - on different platforms, 123
- paths
- extracting from filenames, 234–35
 - specifying relative, 124
- pausing script execution, 428
- percent format, 239
- period (.), used to separate object from property, 53–54
- period (.), used with structures, 53–54
- Pi** (constant), 363
- PICT files, on the Macintosh, 176, 365, 367
- Picture** (statement), 363–65
- picture button controls
- adding to dialog template, 365–67
- picture buttons
- in Dialog Editor, 657
- picture controls
- adding to dialog template, 363–65
 - automatic loading of images into, 180
 - caching, 180
 - deleting image of, 175
 - in Dialog Editor, 657
 - setting image of, 174–76
- picture libraries, creating or modifying, 672–73
- PictureButton** (statement), 365–67
- pictures, specifying, in Dialog Editor, 671–72
- platform constants**, 87
- ebDOS (constant), 87
 - ebMacintosh (constant), 87
 - ebOS2 (constant), 87
 - ebSolaris (constant), 87
 - ebWin16 (constant), 87
 - ebWin32 (constant), 87
- Pmt** (function), 367–68
- Point** (method)
- of WMConstraint, 529
 - of WMDocument, 581
- PopupMenu** (function), 368–69
- portability of compiled code, 122
- PPmt** (function), 369–71
- precedence of operators, 355
- precision
- loss of, 55
 - of operators, 356
- predefined dialogs. *See* dialogs, built-in
- predefined objects, table of, 347
- present value, calculating, 384–85
- Preserve** (keyword), 398–99
- preserving elements while redimensioning arrays, 398–99
- Print** (statement), 371–72
- print zones, 371, 373
- Print#** (statement), 371–72
- printer orientation
- constants used with
 - ebLandscape** (constant), 198
 - ebPortrait** (constant), 203
 - getting, 374–75
 - setting, 375
- PrinterGetOrientation** (function), 374–75
- PrinterSetOrientation** (statement), 375
- PrintFile** (function), 375–76
- printing
- files, 375–76
 - list of language elements, 14–15
 - to **stdout**, 371–72
 - to viewports, 371–72
- Private** (keyword), 246, 449
- Private** (statement), 376–77
- private variables, declaring, 376–77
- procedures. *See* subroutines; functions
- list of language elements, 15
 - tracing calls of, in Script Editor, 639
- program flow. *See* control structures
- promotion
- automatic, 356
 - of operands in expressions, 225
- prompting for input. *See* dialogs, built-in
- Properties, 23
- accessing, 345
 - defined, 344
 - with OLE automation, 343
- Public** (keyword), 246, 449

- Public** (statement), 377–80
- public variables, declaring, 377–80
- push buttons
 - adding to dialog template, 380–81
 - checking for existence of, 92
 - checking if enabled, 91–92
 - getting label of, 177
 - in Dialog Editor, 657
 - selecting, 415
 - setting label of, 176
- PushButton** (statement), 380–81
- Put** (statement), 381–84
- Pv** (function), 384–85
- qualifiers
 - of database owners, 436
 - of databases, 436
 - of tables, 436
- QueEmpty** (statement), 385–86
- QueFlush** (statement), 386
- QueKeyDn** (statement), 386–87
- QueMouseClicked** (statement), 388–89
- QueMouseDbtClk** (statement), 389–90
- QueMouseDbtDn** (statement), 390–91
- QueMouseDn** (statement), 391–92
- QueMouseMove** (statement), 392
- QueMouseMoveBatch** (statement), 393–94
- QueMouseUp** (statement), 394
- QueSetRelativeWindow** (statement), 394–95
- queues
 - constants used with
 - ebLeftButton** (constant), 198–99
 - ebRightButton** (constant), 206
 - emptying, 385–86
 - playing back, 386
 - waiting for playback of, 184
- radians, converting to degrees, 84
- Random** (function), 395
- Random** (keyword), 350–53
- random files
 - opening, 350–53
 - reading, 251–53
 - setting record length, 352
 - writing to, 381–84
- random numbers
 - generating
 - between 0 and 1, 403–4
 - within range, 395
 - initializing random number generator, 395–96
- Randomize** (statement), 395–96
- Rate** (function), 396–97
- Read** (keyword), 350–53
- ReadIni\$** (function), 397
- ReadIniSection** (statement), 397–98
- records. *See* user-defined types
- recursion, 247, 450
- Redim** (statement), 398–99
- redimensioning arrays, 398–99
- reference counting, 344
- regular expressions, with **Like** (operator), 297–98
- relaxed type checking, 63–64
- Rem** (statement), 400
- remainder, calculating, 322
- remote execution, with **DDEExecute** (statement), 138–39
- renaming files, 330–31
- repeating statements. *See* looping
- replacing text, in Script Editor, 634–35
- requesting user input. *See* dialogs, built-in
- reserved words, 291
- Reset** (statement), 400
- resetting error handler, 349
- resizing
 - applications, 75–76
 - controls, in Dialog Editor, 668–69
 - dialog boxes, in Dialog Editor, 668
 - windows, 489–90
- resolution, of screen, 408–9–10
- resources, of operating environment, 454
- restoring
 - applications, 73–74
 - windows, 488–89
- restricted words, 291
- Resume** (statement), 220, 348–50, 400–401
- Return** (statement), 401–2
- Right, Right\$** (functions), 402
- Rmdir** (statement), 403
- Rnd** (function), 403–4
- rounding, 226
- RSet** (statement), 404
- RTrim, RTrim\$** (functions), 405
- running other programs, 426–27
- runtime errors, 699–702
- SaveFilename\$** (function), 406–7
- scientific format, 239
- scoping
 - of constants, 117
 - of object variables, 421
- Screen.DlgBaseUnitsX** (property), 407–8
- Screen.DlgBaseUnitsY** (property), 408
- Screen.Height** (property), 408–9
- Screen.TwipsPerPixelX** (property), 409
- Screen.TwipsPerPixelY** (property), 409

778 Working Model Basic User's Manual

Screen.Width (property), 409–10

Script Editor, 619. *See also* debugging scripts, in Script Editor

- application window, 620

- comments, adding to script, 632

- Debug menu, 649

- dialog box templates, editing for use in, 636

- Edit menu, 647–48

- edit pane, 620

- exiting from, 646

- File menu, 646–47

- Help menu, 649

- Help system, 624

 - getting context-sensitive help, 624–25

 - searching for help on specific topics, 625–26

 - using contents, 626

- insertion point, moving, 626–27

 - to specified line, 627–28

 - with keyboard, 622–23

 - with mouse, 627

- instruction pointer, 638

 - moving to another line in subroutine, 639

- keyboard shortcuts

 - debugging, 623–24

 - editing, 623

 - general, 621–22

 - navigating, 622–23

- pane separator, 620

- Run menu, 648

- scripts

 - checking syntax of, 635–36

 - navigating within, 622–23, 626–28

 - pausing execution of, 637

 - running, 637

 - stopping execution of, 637

 - tracing execution of, 638–39

- selection highlight, 629

- statements, continuing on multiple lines, 633

- status bar, 620

- text

 - copying, 631

 - cutting, 631

 - deleting, 630–31

 - inserting, 628

 - pasting, 631

 - replacing, 634–35

 - searching for, 633–34

 - selecting, 629–30

- toolbar, 620–21

- undoing editing operations, 631–32

- watch pane, 620

Scripting Operation, 21

scripts, debugging, in Script Editor. *See* debugging scripts, in Script Editor

scrolling

- HLine** (statement), 266–67

- HPage** (statement), 267–68

- HScroll** (statement), 268

- VLine** (statement), 479

- VPage** (statement), 479–80

- VScroll** (statement), 480

searching for text, in Script Editor, 633–34

Second (function), 410

seed, for random number generator, 395–96

Seek (function), 410–11

Seek (statement), 411–12

SELECT (SQL statement), 434, 440

Select...Case (statement), 412–13

SelectBox (function), 414–15

SelectButton (statement), 415

SelectComboBoxItem (statement), 416

selecting

- controls, in Dialog Editor, 662

- dialog boxes, in Dialog Editor, 662

selecting text, in Script Editor, 629–30

SelectListBoxItem (statement), 416–17

sending keystrokes, 386–87

SendKeys (statement), 184

separator lines, in dialog boxes, 265

sequential files

- opening, 350–53

- reading, 274–76

- reading lines from, 298–99

- writing to, 371–72, 492–93

Set (statement), 420–21

SetAttr (statement), 421–22

SetCheckBox (statement), 422–23

SetEditText (statement), 423–24

SetOption (statement), 424–25

Sgn (function), 425–26

Shared (keyword), 350–53

sharing

- data. *See* DDE

- files, 352

sharing variables, 379

Shell (function), 64, 426–27

short date format, 240

short time format, 240

showing

- applications, 75

- dialog controls, 180–82

sign, of numbers, 425–26

simulating events. *See* controlling applications

Sin (function), 427

sine function (**Sin**), 427

Single (data type), 427–28

conversion to, 124–25

range of values, 428

storage requirements, 428

Sleep (statement), 428

Sln (function), 429

sounds

Beep (statement), 88

Mci (function), 313–15

Space, Space\$ (functions), 429–30

Spc (function), 371, 373, 430

special characters, 103, 419

escape characters, 358

SQLBind (function), 430

SQLClose (function), 431

SQLError (function), 432

SQLExecQuery (function), 433

SQLGetSchema (function), 435

SQLOpen (function), 438

SQLRequest (function), 439

SQLRetrieve (function), 441

SQLRetrieveToFile (function), 442

Sqr (function), 444

square root function (**Sqr**), 444

standard numeric format, 239

Statements, 24, 34

Static (keyword), 246, 449

status bar

in Dialog Editor, 653

in Script Editor, 620

stdout, printing to, 371–72

Step (keyword), 237–38

Stop (statement), 444

stopping script execution, 213, 444

storage

for fixed-length strings, 447

Str, Str\$ (functions), 445

straight-line depreciation, 429

StrComp (function), 445–46

String (data type), 446–48

string functions

Item\$ (function), 289

LCase, LCase\$ (functions), 293–94

Left, Left\$ (functions), 294

Len (function), 294–96

Line\$ (function), 299–300

LTrim, LTrim\$ (functions), 311

Mid, Mid\$ (functions), 317–18

Option Compare (statement), 356–57

Right, Right\$ (functions), 402

RTrim, RTrim\$ (functions), 405

Space, Space\$ (functions), 429–30

StrComp (function), 445–46

String, String\$ (functions), 448

Trim, Trim\$ (functions), 464

UCase, UCase\$ (functions), 467

Word\$ (function), 490–91

string operators

& (operator), 48

+ (operator), 50–52

Like (operator), 297–98

list of, 15

String, String\$ (functions), 448

strings

comparing, 114, 297–98, 356–57, 445–46

concatenation, 48, 50–52

vs. addition, 48, 51

converting from numbers, 445

converting to, 125

converting to lowercase, 293–94

converting to numbers, 471–72

converting to uppercase, 467

copying, 310–11, 404

counting items within, 289–90

counting lines within, 300–301

counting words within, 491–92

escape characters in, 358

finding one within another, 278–79

fixed-length vs. variable-length, 447

fixed-length, declaring, 161, 376, 378

getting leftmost characters from, 294

getting length of, 294–96

getting rightmost characters from, 402

getting substrings from, 317–18

list of language elements, 15–16

of same characters, 448

of spaces, 429–30

parsing by item, 289

printing, 371–72

reading from sequential files, 274–76–77, 298–99

requesting from user, 82–83, 277–78

retrieving items from, 289

retrieving lines from, 299–300

retrieving words from, 490–91

setting substrings in, 318–19

String (data type), 446–48

trimming leading and trailing spaces from, 464

trimming leading spaces from, 311

trimming trailing spaces from, 405

- writing to sequential files, 371–72, 492–93
- stripping spaces. *See* trimming
- structures. *See* user-defined types
- Sub...End Sub** (statement), 448–51
 - exiting subroutine, 224
- subroutines
 - defining, 448–51
 - exiting subroutine, 224
 - naming conventions of, 449
- substrings
 - finding, 278–79
 - getting, 317–18
 - getting leftmost characters from, 294
 - getting rightmost characters from, 402
 - setting, 318–19
- sum of years' digits depreciation, 452–53
- Switch** (function), 451–52
- SYD** (function), 452–53
- syntax, checking, in Script Editor, 635–36
- System 7.0, 312
- System** calling convention, 145
- system date. *See* dates
- system time. *See* time
- System.Exit** (method), 453
- System.FreeMemory** (property), 453–54
- System.FreeResources** (property), 454
- System.MouseTrails** (method), 454
- System.Restart** (method), 454–55
- System.TotalMemory** (property), 455
- System.WindowsDirectory\$** (property), 455
- System.WindowsVersion\$** (property), 455–56
- Tab** (function), 371, 373, 457
- tables
 - retrieving column data types, 436
 - retrieving column names of, 436
 - retrieving list of, 436
 - retrieving qualifier of, 436
- Tan** (function), 457–58
- tangent function (**Tan**), 457–58
- task list, filling array with, 70–71
- templates. *See* user dialogs
- testing dialog boxes, in Dialog Editor, 668, 677–79
- Text** (statement), 458–59
- text boxes
 - adding to dialog template, 459–61
 - checking existence of, 211–12
 - checking if enabled, 211
 - getting content of, 177, 257–58
 - in Dialog Editor, 657, 679
 - setting content of, 176, 423–24
- text controls
 - adding to dialog template, 458–59
 - getting label of, 177
 - in Dialog Editor, 657
 - setting label of, 176
- TextBox** (statement), 459–61
- thermometers, in message dialogs, 329–30
- tiling desktop windows, 158–59
- time
 - forming from components, 463
 - getting current time, 339–40, 461
 - getting specific time, 463
 - hours, 267
 - minutes, 319–20
 - seconds, 410
 - seconds since midnight, 462
 - setting current time, 461–62
- Time**, **Time\$** (functions), 461
- Time**, **Time\$** (statements), 461–62
- time/date functions. *See* date/time functions
- Timer** (function), 462
- TimeSerial** (function), 463
- TimeValue** (function), 463–64
- toolbar
 - in Dialog Editor, 653–54
 - in Script Editor, 620–21
- trigonometric functions
 - Atn** (function), 84
 - Cos** (function), 118
 - Sin** (function), 427
 - Tan** (function), 457–58
- Trim**, **Trim\$** (functions), 464
- trimming
 - leading and trailing spaces from strings, 464
 - leading spaces from strings, 311
 - trailing spaces from strings, 405
- True** (constant), 464
- true/false format, 240
- truncating numbers, 236–37, 279–80
- twips per pixel, calculating, 409
- Type** (statement), 465–66
- type checking, relaxed, with **Declare** (statement), 63–64
- type coercion, 225
- type-declaration characters
 - effect on interpretation when reading numbers from sequential files, 274
 - for **Currency**, 126
 - for **Double**, 186
 - for **Integer**, 280
 - for **Long**, 310
 - for **Single**, 428
 - for **String**, 447

- used when converting to number, 288
- used when declaring literals, 305–6
- used with **Dim** (statement), 161
- used with external subroutines and functions, 145
- UBound** (function), 466–67
 - used with OLE arrays, 466
- UCase, UCase\$** (functions), 467
- unary minus operator, 52–53
- underflow, 55
- undo
 - in Dialog Editor, 674
 - in Script Editor, 631–32
- uninitialized objects, 343, 345
 - Nothing** (constant), 339
 - testing for with **Is** (operator), 283
- universal date format
 - reading, 275
 - used with literals, 129, 305
 - writing, 492
- Unlock** (statement), 467–69
- unlocking file regions, 467–69
- unsupported language elements, 123
- UPDATE (SQL statement), 434, 440
- uppercasing strings, 467
- user dialogs
 - automatic timeout for, 160
 - available controls in, 88
 - Begin Dialog** (statement), 88–90
 - CheckBox** (statement), 99–101
 - ComboBox** (statement), 109–10
 - control outside bounds of, 172
 - creating, 88–90
 - default button for, 159
 - Dialog** (function), 159–60
 - Dialog** (statement), 160–61
 - dialog procedures of, 171–74
 - DlgControlId** (function), 159–60
 - DlgEnable** (function), 167–68
 - DlgEnable** (statement), 168–69
 - DlgFocus** (function), 169
 - DlgFocus** (statement), 169–70
 - DlgListBoxArray** (function), 170
 - DlgListBoxArray** (statement), 170–71
 - DlgProc** (function), 171–74
 - DlgSetPicture** (statement), 174–76
 - DlgText** (statement), 176–77
 - DlgText\$** (function), 177–78
 - DlgValue** (function), 178
 - DlgValue** (statement), 178–79
 - DlgVisible** (function), 179–80
 - DlgVisible** (statement), 180–82
 - DropListBox** (statement), 186–88
 - expression evaluation within, 90
 - GroupBox** (statement), 265–66
 - idle processing for, 173, 691
 - invoking, 159–60–61
 - list of language elements, 16–17
 - ListBox** (statement), 301–3
 - nesting capabilities of, 174
 - OKButton** (statement), 347–48
 - OptionButton** (statement), 358–59
 - OptionGroup** (statement), 361–62
 - Picture** (statement), 363–65
 - PictureButton** (statement), 365–67
 - pressing Enter within, 348
 - pressing Esc within, 95
 - PushButton** (statement), 380–81
 - required statements within, 89
 - showing, 172
 - Text** (statement), 458–59
 - TextBox** (statement), 459–61
- user-defined errors
 - converting to, 127–28
 - generating, 218–19
 - printing, 371
 - printing to sequential files, 373
 - reading from binary/random files, 252
 - reading from sequential files, 275
 - testing for, 285–86
 - writing to random/binary files, 383
 - writing to sequential files, 492
- user-defined functions. *See* functions
- user-defined types, 470–71
 - copying, 470
 - declaring, 470
 - defining, 465–66
 - getting size of, 294–96, 471
 - passing, 471
- Val** (function), 471–72
- Value** (property), 269–70
- variables
 - assigning objects, 420–21
 - declaring
 - as local, 161–63
 - as private, 376–77
 - as public, 377–80
 - with **Dim**, 161–63
 - with **Private** (statement), 376–77
 - with **Public** (statement), 377–80
 - getting storage size of, 294–96
 - implicit declaration of, 161
 - initial values of, 162, 377, 379

782 Working Model Basic User's Manual

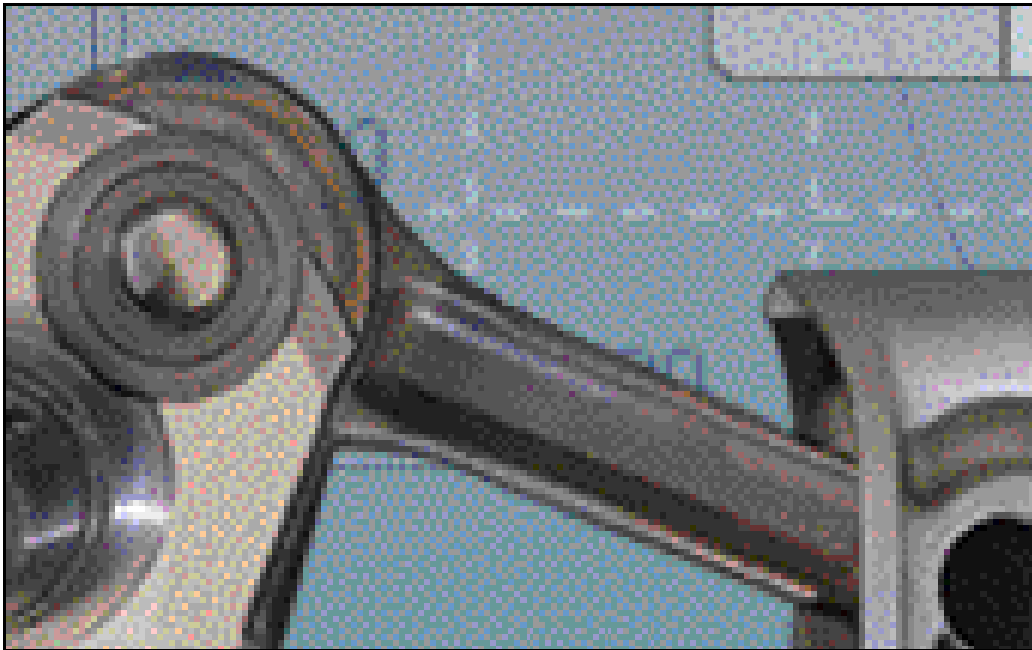
- list of language elements, 17
- naming conventions of, 163
- watch, in Script Editor, 642–45
- Variant** (data type), 472–76
- variants
 - #FALSE#**, 275
 - #NULL#**, 275
 - #TRUE#**, 275
 - adding, 51, 474
 - assigning, 473
 - automatic promotion of, 356
 - containing no data, 342–43, 474
 - converting to, 126–27
 - disadvantages, 475
 - Empty** (constant), 212
 - getting length of, 294–96
 - getting types of, 473, 476–77
 - list of language elements, 17–18
 - Null** (constant), 342–43
 - operations on, 474
 - passing nonvariant data to routines taking variants, 475
 - passing to routines taking nonvariants, 476
 - printing, 371–72
 - reading from sequential files, 274–76
 - storage requirements of, 475
 - testing for **Empty**, 285
 - testing for **Error**, 285–86
 - testing for **Null**, 287
 - testing for objects, 288–89
 - types of, 472, 476
 - ebBoolean** (constant), 191
 - ebCurrency** (constant), 192
 - ebDate** (constant), 193
 - ebDouble** (constant), 195
 - ebEmpty** (constant), 195
 - ebError** (constant), 193
 - ebInteger** (constant), 197–98
 - ebLong** (constant), 199
 - ebNull** (constant), 202
 - ebObject** (constant), 202
 - ebSingle** (constant), 206–7
 - ebString** (constant), 207
 - ebVariant** (constant), 208
- VarType** (function), 476–77
- version
 - of BasicScript, 87–88
 - of Windows, 455–56
- ViewportClear** (statement), 477
- ViewportClose** (statement), 477–78
- ViewportOpen** (statement), 478–79
- viewports
 - clearing, 477
 - closing, 477–78
 - keys used in, 478
 - opening, 478–79
 - printing to, 371–72
- Visual Basic, error messages, 699
- VisualBasic, 20
- VLine** (statement), 479
- VPage** (statement), 479–80
- VScroll** (statement), 480
- wallpaper, changing desktop, 156–57
- watch variables, in Script Editor, 642
 - adding, 642–44
 - deleting, 644
 - modifying value of, 644–45
 - selecting, 644
- waveform audio, **Mci** (function), 313–15
- Weekday** (function), 480–81
- While...Wend** (statement), 481–82
- Width#** (statement), 482
- width, of screen, 409–10
- wildcards. *See also* **MacID** (function)
 - used with **Dir**, **Dir\$** (functions), 164
- win.ini file, 136, 157, 245, 375, 397, 398, 464, 494
- WinActivate** (statement), 482–83
- WinClose** (statement), 483–84
- windows
 - activating, 482–83
 - capturing, 157–58
 - closing, 483–84
 - constants used with
 - ebMaximized** (constant), 199
 - ebMinimized** (constant), 200
 - ebRestored** (constant), 205
 - directory of, 455
 - finding, 484–85
 - getting list of, 485
 - getting value of, 269–70
 - maximizing, 485–86
 - minimizing, 486–87
 - moving, 487–88
 - resizing, 489–90
 - restoring, 488–89
 - scrolling, 266–67–68, 479–80
 - version of, 455–56
- WinFind** (function), 484–85
- WinList** (statement), 485
- WinMaximize** (statement), 485–86

-
- WinMinimize** (statement), 486–87
 - WinMove** (statement), 487–88
 - WinRestore** (statement), 488–89
 - WinSize** (statement), 489–90
 - WM** (constant), 33, 498
 - WM Basic, 33
 - WM.ActiveDocument** (property), 499
 - WM.DeleteMenuItem** (method), 499
 - WM.Documents** (property), 500
 - WM.EnableMenuItem** (method), 501
 - WM.GetMenuItem** (method), 502
 - WM.InsertMenuItem** (method), 503
 - WM.LoadWMBLibrary** (method), 504
 - WM.NewDocument** (method), 505
 - WM.Open** (method), 506
 - WM.RunScript** (method), 506
 - WM.ShowAppearanceWindow** (property), 508
 - WM.ShowGeometryWindow** (property), 508
 - WM.ShowPropertiesWindow** (property), 509
 - WM.UnloadWMBLibrary** (method), 507
 - WM.Version** (property), 509
 - WMBody** (object), 36, 509
 - WMCell** (object), 512
 - WMConstraint** (object), 39, 514
 - WMConstraint.ActiveWhen** (property), 516
 - WMConstraint.ActuatorType** (property), 517
 - WMConstraint.AddVertex** (method), 517
 - WMConstraint.AlwaysActive** (property), 519
 - WMConstraint.AppendPoint** (method), 518
 - WMConstraint.AutoComputeGearRatio** (property), 520
 - WMConstraint.ClosedSlot** (property), 520
 - WMConstraint.CurrentLength** (property), 520
 - WMConstraint.CurrentRotation** (property), 520
 - WMConstraint.DamperK** (property), 521
 - WMConstraint.DeleteVertex** (method), 521
 - WMConstraint.Elasticity** (property), 522
 - WMConstraint.Exponent** (property), 522
 - WMConstraint.Field** (property), 523
 - WMConstraint.FR, FTheta** (properties), 523
 - WMConstraint.FX, FY** (properties), 524
 - WMConstraint.GearRatio** (property), 525
 - WMConstraint.GetVertex** (method), 526
 - WMConstraint.Internal** (property), 526
 - WMConstraint.InternalBody** (property), 526
 - WMConstraint.K** (property), 527
 - WMConstraint.Kind** (property), 528
 - WMConstraint.Length** (property), 528
 - WMConstraint.MotorType** (property), 529
 - WMConstraint.Point** (method), 529
 - WMConstraint.PointCount** (property), 531
 - WMConstraint.RodActive** (property), 531
 - WMConstraint.RodAlwaysActive** (property), 532
 - WMConstraint.RotateWithBody** (properties), 532
 - WMConstraint.Rotation** (property), 532
 - WMConstraint.Torque** (property), 533
 - WMConstraint.VertexCount** (property), 534
 - WMDocument** (object), 34, 534
 - WMDocument.AirResistanceType** (property), 538
 - WMDocument.AirResistanceV2Coeff** (property), 539
 - WMDocument.AirResistanceVCoeff** (property), 539
 - WMDocument.AnimationStep** (property), 540
 - WMDocument.AssemblyError** (property), 540
 - WMDocument.AutoAnimationStep** (property), 541
 - WMDocument.AutoAssemblyError** (property), 541
 - WMDocument.AutoEraseTrack** (property), 542
 - WMDocument.AutoIntegratorError** (property), 542
 - WMDocument.AutoOverlapError** (property), 543
 - WMDocument.AutoSignificantDigits** (property), 543
 - WMDocument.Bodies** (property), 544
 - WMDocument.Body** (method), 544
 - WMDocument.ChargeUnit** (property), 545
 - WMDocument.Close** (method), 546
 - WMDocument.Collide** (method), 546
 - WMDocument.CombineTapeScroll** (property), 547
 - WMDocument.Constraint** (method), 547
 - WMDocument.Constraints** (property), 548
 - WMDocument.ControlsLocked** (property), 549
 - WMDocument.Copy** (method), 549
 - WMDocument.CurrentFrame** (property), 550
 - WMDocument.Cut** (method), 551
 - WMDocument.DecimalDigits** (property), 551
 - WMDocument.DecimalFormat** (property), 551
 - WMDocument.Delete** (method), 552
 - WMDocument.DeletePauseControl** (method), 553
 - WMDocument.DistanceUnit** (property), 554
 - WMDocument.ElectricPotentialUnit** (property), 555
 - WMDocument.ElectrostaticConst** (property), 555
 - WMDocument.ElectroStaticsOn** (property), 556
 - WMDocument.EnergyUnit** (property), 556
 - WMDocument.EraseMeterValues** (method), 557
 - WMDocument.EraseTrack** (method), 558
 - WMDocument.ExportDXF** (method), 558
 - WMDocument.ExportMeterData** (method), 559
 - WMDocument.ExportStartFrame** (property), 559
 - WMDocument.ExportStopFrame** (property), 560
 - WMDocument.ForceFieldFX, ForceFieldFY, ForceFieldT** (property), 560
 - WMDocument.ForceFieldType** (property), 560
 - WMDocument.ForceUnit** (property), 561
 - WMDocument.FrequencyUnit** (property), 562
 - WMDocument.GetPauseControlType** (method), 562

- WMDocument.Gravity** (property), 563
WMDocument.HistoryFrames (property), 564
WMDocument.ImportDXF (method), 564
WMDocument.Input (method), 565
WMDocument.Inputs (property), 566
WMDocument.Integrator (property), 566
WMDocument.IntegratorError (property), 567
WMDocument.Join (method), 568
WMDocument.LinearGravityConst (property), 568
WMDocument.MassUnit (property), 569
WMDocument.Name (property), 569
WMDocument.NewBody (method), 570
WMDocument.NewConstraint (method), 570
WMDocument.NewInput (method), 572
WMDocument.NewOutput (method), 572
WMDocument.NewPauseControl (method), 573
WMDocument.NewPoint (method), 574
WMDocument.Object (method), 575
WMDocument.Objects (property), 575
WMDocument.Output (method), 576
WMDocument.Outputs (property), 577
WMDocument.OverlapError (property), 578
WMDocument.Paste (method), 578
WMDocument.PauseControl (method), 579
WMDocument.PauseControlCount (property), 580
WMDocument.PlanetaryGravityConst (property), 580
WMDocument.PlayerMode (property), 581
WMDocument.Point (method), 581
WMDocument.Points (property), 582
WMDocument.PowerUnit (property), 583
WMDocument.Reset (method), 584
WMDocument.RetainMeterValues (property), 583
WMDocument.RotationalVelocityUnit (property), 584
WMDocument.RotationUnit (property), 585
WMDocument.Run (method), 585
WMDocument.Save (method), 586
WMDocument.SaveAs (method), 587
WMDocument.ScaleFactor (property), 587
WMDocument.ScrollTo (method), 588
WMDocument.Select (method), 588
WMDocument.SelectAll (method), 589
WMDocument.Selection (property), 590
WMDocument.SetPauseControlType (method), 591
WMDocument.ShowCoordinates (property), 592
WMDocument.ShowGridLines (property), 593, 605
WMDocument.ShowHelpRibbon (property), 593
WMDocument.ShowRulers (property), 594
WMDocument.ShowScrollBars (property), 594
WMDocument.ShowTapeControl (property), 595
WMDocument.ShowToolPalette (property), 596
WMDocument.ShowXYAxes (property), 596
WMDocument.SignificantDigits (property), 597
WMDocument.SimulationMode (property), 597
WMDocument.SkipFrames (property), 598
WMDocument.Split (method), 598
WMDocument.StartHere (method), 599
WMDocument.TimeUnit (property), 600
WMDocument.Tracking (property), 600
WMDocument.UnitSystem (property), 601
WMDocument.Update (method), 602
WMDocument.VariableIntegrationStep (property), 603
WMDocument.VelocityUnit (property), 604
WMDocument.WarnInaccurate (property), 605
WMDocument.WarnInconsistent (property), 606
WMDocument.WarnOverlap (property), 607
WMDocument.WarnRedundant (property), 607
WMInput (object), 608
WMObject (object), 610
WMOutput (object), 613
WMOutputColumn (object), 615
WMPoint (object), 37, 616
Word\$ (function), 490–91
WordCount (function), 491–92
word-wrapping, in **MsgBox** (statement), 325
Write (keyword), 350–53
Write# (statement), 492–93
WriteIni (statement), 493–94
Xor (operator), 494–95
Year (function), 495
yes/no format, 239
yielding, 184, 428

Changes to Working Model Basic™

This section describes changes in the WM Basic™ Script language that will affect scripts written for Working Model 3.0.3.



Title: Language differences between BasicScript 2.1, 2.2, and 2.25

Category: BasicScript Language

Document Number: 0000223A

Circulation: Public - Everyone

Applies To: All versions of BasicScript

This document describes the changes in the BasicScript language that affect scripts written for BasicScript 2.1 or BasicScript 2.0. The goal of this document is to describe what causes older scripts to break when running them in BasicScript 2.2.

Other related documents:

- "Upgrading to BasicScript 2.25" describes programmatic changes in the BasicScript APIs that are relevant when upgrading from BasicScript 2.1 to 2.2 and 2.25. This document is distributed with the *BasicScript Programmer's Guide* and is available for download from the Partners Web site.
- "BasicScript 2.1 to 2.2 Changes in Language Reference" described the changes in the Language Reference that occurred between versions 2.1 and 2.2. This document is available for download from the Partners Web site.
- "Modifying compilation for backward compatibility" is a knowledge base article describing special undocumented flags for compiling old scripts. The knowledge base is available for browsing/searching on the Partners Web site.

SendKeys

The default value for the second parameter to Shell was changed from True to False. Script that contain statement such as:
SendKeys "Hello"

Will need to be changed to:
SendKeys "Hello",True

Type Declaration Characters

In version 2.1, the following was allowed:
Dim a\$ As String

This now produces an error. The following are correct syntaxes:
Dim a\$
Dim a As String

(Note: The embedder can disable this new behavior by setting the CF_RESERVED2 flag in the wFlags member of the ebCOMPILESTRUCT structure.)

Using Keywords as Identifiers

In BasicScript 2.1, the following was allowed:
MsgBox% = 10

In BasicScript 2.2, this is not allowed, as MsgBox is a built-in keyword. You can get around this by renaming the variable to a different unreserved name or by declaring the variable with the Dim, Private, or Public statement:

```
Dim MsgBox As Integer
MsgBox = 10
```

Once redeclared in this manner, the MsgBox command/function can no longer be called to display a dialog box.

The re-use of language elements in this manner only applies to built-in keywords defined either by BasicScript or by any of your application's extensions. Additional restrictions apply to reserved words (a subcategory of keywords as described below).

(Note: The embedder can disable this new behavior by setting the CF_RESERVED1 flag in the wFlags member of the ebCOMPILESTRUCT structure.)

Using Reserved Words as Identifiers

BasicScript 2.1 allowed reserved words to be used as variable names when accompanied by a type declaration character as shown below:

```
For$ = "Hello"
```

This is a very bad practice that should be discouraged. These variables should be renamed in the script.

A complete list of reserved words is contained in the User's Guide in the alphabetical entry entitled "Keywords (topic)".

(Note: The embedder can allow reserved words to be redefined in this manner by setting the CF_RESERVED1 flag in the wFlags member of the ebCOMPILESTRUCT structure.)

Default Data Type

In version 2.0 of BasicScript, the default data type was Integer. BasicScript 2.1 and 2.2 now use Variant as the default data type.

This introduces many problems if a script contains many implicit variables and function. In such cases, the following statement can be placed at the top of such scripts to change the default data type back to Integer:

```
Option Default Integer
```

A better solution would be to add an Option Explicit statement at the top of the script to expose all the implicit declarations, then explicitly define each variable and function as the appropriate type. Implicit declarations occasionally cause inadvertent bugs in the script.

Using Predefined Objects as Identifiers

As is the case in all versions of BasicScript, you cannot use the names of predefined objects as identifiers in scripts. For example, if your application introduces an object called "Form" and a predefined object of that type called "Form1", then the user is prevented from using "Form1" as the name of a variable in the script as shown below:

```
Dim Form1 As Integer '-- This will produce an error
```

There are many predefined objects built-in to BasicScript, some of which are new, resulting in potential problems in scripts which use these names for variables, subroutines, or functions. For example, since MsgBox is a new built-in object, the following code will no longer compile:

```
Dim MsgBox As String
MsgBox = "Hello"
```

The following lists all the objects predefined by BasicScript:

Object Name	EB20	EB21	EB22
HWND	Yes	Yes	Yes
Viewport	No	No	Yes
Msg	No	No	Yes
Clipboard	Yes	Yes	Yes
Desktop	Yes	Yes	Yes
System	Yes	Yes	Yes
Net	Yes	Yes	Yes
Screen	Yes	Yes	Yes
Basic	Yes	Yes	Yes
MacID	Yes	Yes	Yes
Err	No	No	Yes

Obsolete Syntax

Although some language elements that have been replaced with more modern syntax, the older style syntax is still supported. The following table shows these replacements in the case that the user wants to use the more modern language element:

Language Element	Replace With
Err (function)	Err[.Number] (default property)
Err (statement)	Err[.Number] (default property)
MsgClose (statement)	Msg.Close (method)
MsgOpen (statement)	Msg.Open (method)
MsgSetText (statement)	Msg.Text (property)
MsgSetThermometer (statement)	Msg.Thermometer (property)
ViewportClear (statement)	Viewport.Clear (method)
ViewportClose (statement)	Viewport.Close (method)
ViewportOpen (statement)	Viewport.Open (method)

New Constants

As is the case in all versions of BasicScript, you cannot use the names of constants as identifiers in scripts. For example, each of the following statements is illegal:

```
Dim ebYes As Integer '<-- Illegal (cannot redefine constant)
ebYes = 10 '<-- Illegal (cannot assign to constant)
```

BasicScript 2.2 introduces a number of constants not present in earlier version. Each of these constants is preceded by the two letters "eb" and are described in the following table:

```
ebAlias ebArray ebBack
ebCFBitmap ebCFDIB ebCFMetafile
ebCFPalette ebCFText ebCFUnicodeText
ebCr ebCrLf ebFirstFourDays
ebFirstFullWeek ebFirstJan1 ebFormFeed
```



```
ebFriday ebFromUnicode ebHide
ebHiragana ebIMEAlphaDb1 ebIMEAlphaSng
ebIMEDisabled ebIMEHiragana ebIMEKatakanaDb1
ebIMEKatakanaSng ebIMENoOp ebIMEOff
ebIMEOn ebKatakana ebLf
ebLINUX ebLowerCase ebMaximizedFocus
ebMinimizedFocus ebMinimizedNoFocus ebMonday
ebNarrow ebNormalFocus ebNormalNoFocus
ebNullChar ebNullString ebProperCase
ebSaturday ebSunday ebTab
ebThursday ebTuesday ebUnicode
ebUpperCase ebUseSystem ebVerticalTab
ebWednesday ebWide
```

Despite the fact that these constants are made unique by their two-letter prefix, there is still a possibility that a duplicate identifier exists in older scripts. In this case, the only alternative is to rename the identifier.

Null Keyword

In BasicScript 2.0, the Null keyword was implemented as a function that returned a zero-length string. In BasicScript 2.1 and above, "Null" is a reserved compiler keyword representing a special state of Variants. Code that uses the Null function must be replaced with an empty string literal as shown below:

The statement:

```
s$ = Null()
```

Should be replaced with:

```
s$ = ""
```

Passing Uninitialized Strings to External Routines

In BasicScript 2.1 and earlier, uninitialized strings were passed to the external routines (i.e., those declared with the Declare statement) as zero-length strings. BasicScript 2.2 and above pass uninitialized strings as NULL.

Resetting the Error State

In BasicScript 2.1 and earlier, the error state was reset when Err was set to -1. In BasicScript 2.2 and later, the error state is reset under the following conditions:

- When a Resume statement is executed.
- When Err is set to -1.
- When an On Error statement is executed.
- When Err.Clear is executed.
- When Exit Sub, Exit Function, End Sub, or End Function is executed.

Instr

In BasicScript 2.1 and earlier, the Instr function was able to determine its behavior from the type of data passed as the first parameter. For example, the following was legal:

```
pos = Instr("Hello, world.", "world", 0)
```

In BasicScript 2.2 and later, the first parameter must be specified in order for the last parameter to be specified. In other words, given the following syntax, Instr uses the rules described below:

Instr([*start*,] *search*,*find*, [,*compare*])

If there are this many parameters	Then Instr assumes the following were specified
2	<i>search</i> , <i>find</i>
3	<i>start</i> , <i>search</i> , <i>find</i>
4	<i>start</i> , <i>search</i> , <i>find</i> , <i>compare</i>

Double Precision in Loop Counters

Consider the following loop:

```
Dim d As Double
For d = 3.7 To 4.1 Step 0.1
MsgBox d
Next d
```

In the above script, the following are printed in the dialogs:

```
3.7
3.8
3.9
4
```

In BasicScript 2.2 and later, the dialog containing "4.1" is not displayed, whereas in version 2.1, the number "4.1" was displayed.

This difference is due to the imprecise internal representation of 0.1 in IEEE format. In the last iteration through the loop, this error has accumulated such that the loop counter is slightly greater than 4.1, preventing the final iteration of the loop. In BasicScript 2.1, the addition was handled differently.

The behavior of BasicScript 2.2 is consistent with that of Visual Basic.