Visual Basic 6.0 Developer Reference

Category	Keywords
Array handling	Array ,Dim, Private, Public, ReDim, IsArray,Erase LBound, UBound
Assignments	Set
Comments	Comments using ' or Rem
Constants/Literals	Empty, Nothing, Null True, False
Control flow	Do...Loop ,For...Next For Each...Next If...Then...Else Select Case While...Wend
Conversions	Abs, Asc, AscB, AscW, Chr, ChrB, ChrW, CBool, CByte,CCur, CDate, CDbl, CInt,CLng, CSng, CStr DateSerial, DateValue,Hex, Oct ,Fix, Int ,Sgn TimeSerial, TimeValue
Dates/Times	Date, Time ,DateAdd, DateDiff, DatePart,DateSerial, DateValue,Day, Month, MonthName, Weekday, WeekdayName, Year,Hour, Minute, Second, Now TimeSerial, TimeValue
Declarations	Const, Dim, Private, Public, ReDim,Function, Sub
Formatting Strings	FormatCurrency,FormatDateTime,FormatNumber FormatPercent
Error Handling	On Error, Err
Input/Output	InputBox ,LoadPicture,MsgBox
Literals	Empty, False,Nothing ,Null,True
Math	Atn, Cos, Sin, Tan, Exp, Log, Sqr,Randomize, Rnd
Miscellaneous	RGB Function
Objects	CreateObject, Dictionary,Drive Object, Drives Collection, Err File Object, Files Collection, FileSystemObject, Folder Object,Folders Collection, GetObject, TextStream
Operators	Addition (+), Subtraction (-), Exponentiation (^),Modulus arithmetic (Mod) ,Multiplication (*), Division (/), Integer Division (\), Negation (-),String concatenation (&) ,Equality (=), Inequality (<>), Less Than (<), Less Than or Equal To (<=), Greater Than (>), Greater Than or Equal To (>=) ,Is ,And, Or, Xor,Eqv, Imp
Options	Option Explicit
Procedures	Call ,Function, Sub
Rounding	Abs ,Int, Fix, Round ,Sgn
Script Engine ID	ScriptEngine, ScriptEngineBuildVersion, ScriptEngineMajorVersion,ScriptEngineMinorVersion
Strings	Asc, AscB, AscW, Chr, ChrB, ChrW ,Filter, InStr, InStrB, InStrRev,Join,Len, LenB,LCase, UCase, Left, LeftB,Mid, MidB,Right, RightB,Replace, Space, Split, StrComp, String,StrReverse, LTrim, RTrim, Trim
Variants	IsArray, IsDate, IsEmpty,IsNull ,IsNumeric,IsObject, TypeName, VarType

Visual Basic for Applications Features not in VBScript

Category	Omitted Feature/Keyword
Array Handling	Option Base, Declaring arrays with lower bound <> 0
Collection	Add, Count, Item, Remove, Access to collections using ! character (e.g., MyCollection!Foo)
Conditional Compilation	#Const , #If...Then...#Else
Control Flow	DoEvents, GoSub...Return, GoTo, On Error GoTo, On...GoSub, On...GoTo ,Line numbers, Line labels, With...End With
Conversion	CVar, CVDate, Str, Val
Data Types	All intrinsic data types except Variant, Type...End Type
Date/Time	Date statement, Time statement, Timer
DDE	LinkExecute, LinkPoke, LinkRequest, LinkSend
Debugging	Debug.Print, End, Stop
Declaration	Declare (for declaring DLLs),New, Optional,ParamArray,Property Get, Property Let, Property Set,Static
Error Handling	Erl ,Error, On Error...Resume, Resume, Resume Next
File Input/Output	All traditional Basic file I/O
Financial	All financial functions
Object Manipulation	TypeOf
Objects	Clipboard Collection
Operators	Like
Options	Def*type,* Option Base,Option Compare,Option Private Module
Select Case	Expressions containing **Is** keyword or any comparison operators Expressions containing a range of values using the **To** keyword.
Strings	Fixed-length strings, LSet, RSet, Mid Statement StrConv
Using Objects	Collection access using !

VBScript Features not in Visual Basic for Applications

Category	Feature/Keyword
Formatting strings	FormatCurrency ,FormatDateTime, FormatNumber ,FormatPercent, MonthName, WeekdayName
Intrinsic constants	vbGeneralDate, vbLongDate, vbLongTime, vbShortDate ,vbLongDate, vbTristateFalse ,vbTristateMixed, vbTristateTrue ,vbTristateUseDefault
Objects	Dictionary, FileSystemObject,TextStream
Rounding	Round
Strings	Filter, InstrRev,Join,Replace,Split,StrReverse
Script Engine Identification	ScriptEngine, ScriptEngineBuildVersion, ScriptEngineMajorVersion, ScriptEngineMinorVersion

Alphabetical keyword

Abs Function

Description: Returns the absolute value of a number.

Syntax: Abs (number)
The number argument can be any valid numeric expression. If number contains Null, Null is returned; if it is a nun initialized variable, zero is returned.

Remarks: The absolute value of a number is its unsigned magnitude. For example, Abs (-1) and Abs (1) both return 1.

Add Method
Description: Adds a key and item pair to a Dictionary object.
Syntax: ***object*** .Add ***key, item***
The **Add** method has the following parts:

Part	Description
Object	Required. Always the name of a **Dictionary** object.
Key	Required. The *key* associated with the *item* being added.
Item	Required. The *item* associated with the *key* being added.

Remarks: An error occurs if the *key* already exists.

Description: Adds a key and item pair to a Dictionary object.
Syntax: object.Add key, item
The Add method has the following parts:
Part Description: object Required Always the name of a Dictionary object. key Required. The key associated with the item being added. item Required. The item associated with the key being added.

Remarks : An error occurs if the key already exists.

AddFolders Method

Description: **Adds a new** Folder **to a** Folders **collection.**
Syntax: ***object***.AddFolders ***folderName*** **The** AddFolders **method has the following parts:**

Part	Description
object	Required. Always the name of a **Folders** collection.
folderName	Required. The name of the new **Folder** being added.

Remarks
An error occurs if the *folderName* already exists.
Description : Adds a new Folder to a Folders collection.
Syntax: object.AddFolders folderName

The AddFolders method has the following parts:

Part Description :
object Required. Always the name of a Folders collection.
folderName Required. The name of the new Folder being added.

Remarks: An error occurs if the folderName already exists.

Addition Operator (+)

Description: Used to sum two numbers.
Syntax
result **=** *expression1* **+** *expression2*
The **+** operator syntax has these parts:

Part	Description
result	Any numeric variable.
expression1	Any expression.
expression2	Any expression.

Remarks
Although you can also use the **+** operator to concatenate two character strings, you should use the **&** operator for concatenation to eliminate ambiguity and provide self-documenting code.
When you use the **+** operator, you may not be able to determine whether addition or string concatenation will occur.
The underlying subtype of the expressions determines the behavior of the **+** operator in the following way:

If	Then
Both expressions are numeric	Add.
Both expressions are strings	Concatenate.
One expression is numeric and the other is a string	Add.

If one or both expressions are **Null** expressions, *result* is **Null**. If both expressions are **Empty**, *result* is an **Integer** subtype. However, if only one expression is **Empty**, the other expression is returned unchanged as *result*.

Description : Used to sum two numbers.

Syntax : result = expression1 + expression2
The + operator syntax has these parts:

Part Description
result Any numeric variable. (expression1 Any expression) + (expression2 Any expression.)

Remarks: Although you can also use the + operator to concatenate two character strings, you should use the & operator for concatenation to eliminate ambiguity and provide self-documenting code. When you use the + operator, you may not be able to determine whether addition or string concatenation will occur. The underlying subtype of the expressions determines the behavior of the + operator in the following way:

And Operator
Description
Used to perform a logical conjunction on two expressions.
Syntax
result **=** *expression1* **And** *expression2*
The **And** operator syntax has these parts:

Part	Description
result	Any numeric variable.
expression1	Any expression.
expression2	Any expression.

Remarks

If, and only if, both expressions evaluate to **True**, *result* is **True**. If either expression evaluates to **False**, *result* is **False**. The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	The *result* is
True	**True**	**True**
True	**False**	**False**
True	**Null**	**Null**
False	**True**	**False**
False	**False**	**False**
False	**Null**	**False**
Null	**True**	**Null**
Null	**False**	**False**
Null	**Null**	**Null**

The **And** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	The *result* is
0	0	0
0	1	0
1	0	0
1	1	1

Array Function

Description

Returns a **Variant** containing an array.

Syntax

Array(*arglist***)**

The required *arglist* argument is a comma-delimited list of values that are assigned to the elements of an array contained with the **Variant**. If no arguments are specified, an array of zero length is created.

Remarks

The notation used to refer to an element of an array consists of the variable name followed by parentheses containing an index number indicating the desired element. In the following example, the first statement creates a variable named A. The second statement assigns an array to variable A. The last statement assigns the value contained in the second array element to another variable.

```
Dim A
A = Array(10,20,20)
B = A(2).
```

Note A variable that is not declared as an array can still contain an array. Although a **Variant** variable containing an array is conceptually different from an array variable containing **Variant** elements, the array elements are accessed in the same way.

Asc Function

Description

Returns the ANSI character code corresponding to the first letter in a string.

Syntax

Asc(*string***)**

The *string* argument is any valid string expression. If the *string* contains no characters, a run-time error occurs.

Note: The AscB function is used with byte data contained in a string. Instead of returning the character code for the first character, AscB returns the first byte. AscW is provided for 32-bit platforms that use Unicode characters. It returns the Unicode (wide) character code, thereby avoiding the conversion from Unicode to ANSI.

Assignment Operator (=)

Description

Used to assign a value to a variable or property.

Syntax

variable **=** *value*

The **=** operator syntax has these parts:

Part	Description
variable	Any variable or any writable property.
value	Any numeric or string literal, constant, or expression.

Remarks

The name on the left side of the equal sign can be a simple scalar variable or an element of an array. Properties on the left side of the equal sign can only be those properties that are writable at run time.

AtEndOfLine Property

Description

Read-only property that returns **True** if the file pointer immediately precedes the end-of-line marker in a **TextStream** file; **False** if it is not.

Syntax

*object***.AtEndOfLine**

The *object* is always the name of a **TextStream** object.

Remarks

The **AtEndOfLine** property applies only to **TextStream** files that are open for reading; otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfLine** property:

```
Dim fs, a, retstring
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.OpenTextFile("c:\testfile.txt", ForReading, False)
Do While a.AtEndOfLine <> True
   retstring = a.Read(1)
Loop
a.Close
```

AtEndOfStream Property

Description

Read-only property that returns **True** if the file pointer is at the end of a **TextStream** file; **False** if it is not.

Syntax

*object***.AtEndOfStream**

The *object* is always the name of a **TextStream** object.

Remarks

The **AtEndOfStream** property applies only to **TextStream** files that are open for reading, otherwise, an error occurs.

The following code illustrates the use of the **AtEndOfStream** property:

```
Dim fs, a, retstring
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.OpenTextFile("c:\testfile.txt", ForReading, False)
Do While a.AtEndOfStream <> True
   retstring = a.ReadLine
   ...
Loop
a.Close
```

Atn Function

Description

Returns the arctangent of a number.

Syntax

Atn(*number***)**

The *number* argument can be any valid numeric expression.

Remarks

The **Atn** function takes the ratio of two sides of a right triangle (*number*) and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

The range of the result is -pi/2 to pi/2 radians.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Note **Atn** is the inverse trigonometric function of **Tan**, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse **Atn** with the cotangent, which is the simple inverse of a tangent (1/tangent).

Attributes Property

Description

Sets or returns the attributes of files or folders. Read/write or read-only, depending on the attribute.

Syntax

object.**Attributes** [= *newattributes*]

The **Attributes** property has these parts:

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
newattributes	Optional. If provided, *newattributes* is the new value for the attributes of the specified *object*.

Settings

The *newattributes* argument can have any of the following values or any logical combination of the following values:

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file. Attribute is read/write.
Hidden	2	Hidden file. Attribute is read/write.
System	4	System file. Attribute is read/write.

Volume	8	Disk drive volume label. Attribute is read-only.
Directory	16	Folder or directory. Attribute is read-only.
Archive	32	File has changed since last backup. Attribute is read/write.
Alias	64	Link or shortcut. Attribute is read-only.
Compressed	128	Compressed file. Attribute is read-only.

Remarks

The following code illustrates the use of the **Attributes** property with a file:

```
Sub SetClearArchiveBit(filespec)
  Dim fs, f, r
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.GetFile(fs.GetFileName(filespec))
  If f.attributes and 32 Then
      r = MsgBox("The Archive bit is set, do you want to clear it?", vbYesNo,
"Set/Clear Archive Bit")
     If r = vbYes Then
        f.attributes = f.attributes - 32
        MsgBox "Archive bit is cleared."
     Else
        MsgBox "Archive bit remains set."
     End If
  Else
      r = MsgBox("The Archive bit is not set. Do you want to set it?", vbYesNo,
"Set/Clear Archive Bit")
     If r = vbYes Then
        f.attributes = f.attributes + 32
        MsgBox "Archive bit is set."
     Else
        MsgBox "Archive bit remains clear."
     End If
  End If
End Sub
```

AvailableSpace Property

Description

Returns the amount of space available to a user on the specified drive or network share.

Syntax

object.**AvailableSpace**

The *object* is always a **Drive** object.

Remarks

The value returned by the **AvailableSpace** property is typically the same as that returned by the **FreeSpace** property. Differences may occur between the two for computer systems that support quotas. The following code illustrates the use of the **AvailableSpace** property:

```
Sub ShowAvailableSpace(drvPath)
   Dim fs, d, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(fs.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & d.VolumeName  & vbCrLf
   s = s & "Available Space: " & FormatNumber(d.AvailableSpace/1024, 0)
   s = s & " Kbytes"
   MsgBox s
End Sub
```

BuildPath Method

Description

Appends a name to an existing path.

Syntax

object.**BuildPath**(*path*, *name*)

The **BuildPath** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
path	Required. Existing path to which *name* is appended. Path can be absolute or relative and need not specify an existing folder.
name	Required. Name being appended to the existing *path*.

Remarks

The **BuildPath** method inserts an additional path separator between the existing path and the name, only if necessary.

Call Statement

Description

Transfers control to a **Sub** procedure or **Function** procedure.

Syntax

[**Call**] *name* [*argumentlist*]

The **Call** statement syntax has these parts:

Part	Description
Call	Optional keyword. If specified, you must enclose *argumentlist* in parentheses. For example: Call MyProc(0)
name	Required. Name of the procedure to call.
argumentlist	Optional. Comma-delimited list of variables, arrays, or expressions to pass to the procedure.

Remarks

You are not required to use the **Call** keyword when calling a procedure. However, if you use the **Call** keyword to call a procedure that requires arguments, *argumentlist* must be enclosed in parentheses. If you omit the **Call** keyword, you also must omit the parentheses around *argumentlist*. If you use either **Call** syntax to call any intrinsic or user-defined function, the function's return value is discarded.

CBool Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Boolean**.

Syntax

CBool(*expression*)

The *expression* argument is any valid expression.

Remarks

If *expression* is zero, **False** is returned; otherwise, **True** is returned. If *expression* can't be interpreted as a numeric value, a run-time error occurs.

CByte Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Byte**.

Syntax

CByte(*expression*)

The *expression* argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CByte** to force byte arithmetic in cases where currency, single-precision, double-precision, or integer arithmetic normally would occur.

Use the **CByte** function to provide internationally aware conversions from any other data type to a **Byte** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

Note The **ChrB** function is used with byte data contained in a string. Instead of returning a character, which may be one or two bytes, **ChrB** always returns a single byte. **ChrW** is provided for 32-bit platforms that use Unicode characters. Its argument is a Unicode (wide) character code, thereby avoiding the conversion from ANSI to Unicode.

CInt Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Integer**.

Syntax

CInt(*expression*)

The *expression* argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CInt** or **CLng** to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the **CInt** function to provide internationally aware conversions from any other data type to an **Integer** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *expression* lies outside the acceptable range for the **Integer** subtype, an error occurs.

Note **CInt** differs from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the **CInt** function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

Clear Method

Description

Clears all property settings of the **Err** object.

Syntax

object.**Clear**

The object is always the **Err** object.

Remarks

Use **Clear** to explicitly clear the **Err** object after an error has been handled. This is necessary, for example, when you use deferred error handling with **On Error Resume Next**. VBScript calls the **Clear** method automatically whenever any of the following statements is executed:

- **On Error Resume Next**
- **Exit Sub**
- **Exit Function**

CLng Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Long**.

Syntax

CLng(*expression*)

The *expression* argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CInt** or **CLng** to force integer arithmetic in cases where currency, single-precision, or double-precision arithmetic normally would occur.

Use the **CLng** function to provide internationally aware conversions from any other data type to a **Long** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *expression* lies outside the acceptable range for the **Long** subtype, an error occurs.

Note **CLng** differs from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. When the fractional part is exactly 0.5, the **CLng** function always rounds it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2.

Close Method

Description

Closes an open **TextStream** file.

Syntax

object.**Close**

If *expression* lies outside the acceptable range for the **Byte** subtype, an error occurs.

CCur Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Currency**.

Syntax

CCur(*expression*)

The *expression* argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CCur** to force currency arithmetic in cases where integer arithmetic normally would occur.

You should use the **CCur** function to provide internationally aware conversions from any other data type to a **Currency** subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.

CDate Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Date**.

Syntax

CDate(*date*)

The *date* argument is any valid date expression.

Remarks

Use the **IsDate** function to determine if *date* can be converted to a date or time. **CDate** recognizes date literals and time literals as well as some numbers that fall within the range of acceptable dates. When converting a number to a date, the whole number portion is converted to a date. Any fractional part of the number is converted to a time of day, starting at midnight.

CDate recognizes date formats according to the locale setting of your system. The correct order of day, month, and year may not be determined if it is provided in a format other than one of the recognized date settings. In addition, a long date format is not recognized if it also contains the day-of-the-week string.

CDbl Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Double**.

Syntax

CDbl(*expression*)

The *expression* argument is any valid expression.

Remarks

In general, you can document your code using the subtype conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CDbl** or **CSng** to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the **CDbl** function to provide internationally aware conversions from any other data type to a **Double** subtype. For example, different decimal separators and thousands separators are properly recognized depending on the locale setting of your system.

Chr Function

Description

Returns the character associated with the specified ANSI character code.

Syntax

Chr(*charcode*)

The *charcode* argument is a number that identifies a character.

Remarks

Numbers from 0 to 31 are the same as standard, nonprintable ASCII codes. For example, **Chr**(10) returns a linefeed character.

The *object* is always the name of a **TextStream** object.

Column Property

Description

Read-only property that returns the column number of the current character position in a **TextStream** file.

Syntax

object.**Column**

The *object* is always the name of a **TextStream** object.

Remarks

After a newline character has been written, but before any other character is written, **Column** is equal to 1.

CompareMode Property

Description

Sets and returns the comparison mode for comparing string keys in a **Dictionary** object.

Syntax

object.**CompareMode**[= *compare*]

The **CompareMode** property has the following parts:

Part	Description
object	Required. Always the name of a **Dictionary** object.
compare	Optional. If provided, *compare* is a value representing the comparison mode used by functions such as **StrComp**.

Settings

The *compare* argument has the following settings:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Remarks

Values greater than 2 can be used to refer to comparisons using specific Locale IDs LCID). An error occurs if you try to change the comparison mode of a Dictionary object that already contains data. The CompareMode property uses the same values as the *compare* argument for the StrComp function.

Concatenation Operator (&)

Description

Used to force string concatenation of two expressions.

Syntax

result = *expression1* & *expression2*

The **&** operator syntax has these parts:

Part	Description
result	Any variable.
expression1	Any expression.
expression2	Any expression.

Remarks

Whenever an *expression* is not a string, it is converted to a **String** subtype. If both expressions are **Null**, *result* is also **Null**. However, if only one *expression* is **Null**, that expression is treated as a zero-length string ("") when concatenated with the other expression. Any expression that is **Empty** is also treated as a zero-length string.

Const Statement

Description

Declares constants for use in place of literal values.

Syntax

[**Public** | **Private**] **Const** *constname = expression*
The **Const** statement syntax has these parts:

Part	Description
Public	Optional. Keyword used at script level to declare constants that are available to all procedures in all scripts. Not allowed in procedures.
Private	Optional. Keyword used at script level to declare constants that are available only within the script where the declaration is made. Not allowed in procedures.
constname	Required. Name of the constant; follows standard variable naming conventions.
expression	Required. Literal or other constant, or any combination that includes all arithmetic or logical operators except **Is**.

Remarks

Constants are public by default. Within procedures, constants are always private; their visibility can't be changed. Within a script, the default visibility of a script-level constant can be changed using the **Private** keyword.

To combine several constant declarations on the same line, separate each constant assignment with a comma. When constant declarations are combined in this way, the **Public** or **Private** keyword, if used, applies to all of them.

You can't use variables, user-defined functions, or intrinsic VBScript functions (such as **Chr**) in constant declarations. By definition, they can't be constants. You also can't create a constant from any expression that involves an operator, that is, only simple constants are allowed. Constants declared in a **Sub** or **Function** procedure are local to that procedure. A constant declared outside a procedure is defined throughout the script in which it is declared. You can use constants anywhere you can use an expression.

Note Constants can make your scripts self-documenting and easy to modify. Unlike variables, constants can't be inadvertently changed while your script is running.

Copy Method

Description

Copies a specified file or folder from one location to another.

Syntax

object.**Copy** *destination*[, *overwrite*]

The **Copy** method syntax has these parts:

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
destination	Required. Destination where the file or folder is to be copied. Wildcard characters are not allowed.
overwrite	Optional. Boolean value that is **True** (default) if existing files or folders are to be overwritten; **False** if they are not.

Remarks

The results of the **Copy** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.CopyFile** or **FileSystemObject.CopyFolder** where the file or folder referred to by *object* is passed as an argument. You should note, however, that the alternative methods are capable of copying multiple files or folders.

CopyFile Method

Description

Copies one or more files from one location to another.

Syntax

object.**CopyFile** *source*, *destination*[, *overwrite*]

The **CopyFile** method syntax has these parts:

Part	Description
object	Required. The *object* is always the name of a **FileSystemObject**.
source	Required. Character string file specification, which can include wildcard characters, for one or more files to be copied.
destination	Required. Character string destination where the file or files from *source* are to be copied. Wildcard characters are not allowed.
overwrite	Optional. Boolean value that indicates if existing files are to be overwritten. If **True**, files are overwritten; if **False**, they are not. The default is **True**. Note that **CopyFile** will fail if *destination* has the read-only attribute set, regardless of the value of *overwrite*.

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
FileSystemObject.CopyFile "c:\mydocuments\letters\*.doc", "c:\tempfolder\"
```

But you can't use:

```
FileSystemObject.CopyFile "c:\mydocuments\*\R1???97.xls", "c:\tempfolder"
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching files. Otherwise, *destination* is assumed to be the name of a file to create. In either case, three things can happen when an individual file is copied.

- If *destination* does not exist, *source* gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs if *overwrite* is **False**. Otherwise, an attempt is made to copy *source* over the existing file.
- If *destination* is a directory, an error occurs.

An error also occurs if a *source* using wildcard characters doesn't match any files. The **CopyFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs.

CopyFolder Method

Description

Recursively copies a folder from one location to another.

Syntax

object.**CopyFolder** *source*, *destination*[, *overwrite*]

The **CopyFolder** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
source	Required. Character string folder specification, which can include wildcard characters, for one or more folders to be copied.
destination	Required. Character string destination where the folder and subfolders from *source* are to be copied. Wildcard characters are not allowed.
overwrite	Optional. Boolean value that indicates if existing folders are to be overwritten. If **True**, files are overwritten; if **False**, they are not. The default is **True**.

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

```
FileSystemObject.CopyFolder "c:\mydocuments\letters\*", "c:\tempfolder\"
```

But you can't use:

```
FileSystemObject.CopyFolder "c:\mydocuments\*\*", "c:\tempfolder\"
```

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching folders and subfolders. Otherwise,

destination is assumed to be the name of a folder to create. In either case, four things can happen when an individual folder is copied.

- If *destination* does not exist, the *source* folder and all its contents gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an attempt is made to copy the folder and all its contents. If a file contained in *source* already exists in *destination*, an error occurs if *overwrite* is **False**. Otherwise, it will attempt to copy the file over the existing file.
- If *destination* is a read-only directory, an error occurs if an attempt is made to copy an existing read-only file into that directory and *overwrite* is **False**.

An error also occurs if a *source* using wildcard characters doesn't match any folders.

The **CopyFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before an error occurs.

Cos Function

Description

Returns the cosine of an angle.

Syntax

Cos(*number***)**

The *number* argument can be any valid numeric expression that expresses an angle in radians.

Remarks

The **Cos** function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Count Property

Description

Returns the number of items in a collection or **Dictionary** object. Read-only.

Syntax

object.**Count**

The *object* is always the name of one of the items in the Applies To list.

Remarks

The following code illustrates use of the **Count** property:

```
Dim a, d, i             'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"     'Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.Keys              'Get the keys
For i = 0 To d.Count -1 'Iterate the array
   Print a(i)           'Print key
Next
```

CreateFolder Method

Description

Creates a folder.

Syntax

object.**CreateFolder(***foldername***)**

The **CreateFolder** method has these parts:

Part	Description

object	Required. Always the name of a **FileSystemObject**.
foldername	Required. String expression that identifies the folder to create.

Remarks

An error occurs if the specified folder already exists.

CreateObject Function

Description

Creates and returns a reference to an Automation object.

Syntax

CreateObject(*class***)**

The *class* argument uses the syntax *servername.typename* and has these parts:

Part	Description
servername	The name of the application providing the object.
typename	The type or class of the object to create.

Remarks

Automation servers provide at least one type of object. For example, a word-processing application may provide an application object, a document object, and a toolbar object.

To create an Automation object, assign the object returned by **CreateObject** to an object variable:

```
Dim ExcelSheet
Set ExcelSheet = CreateObject("Excel.Sheet")
```

This code starts the application creating the object (in this case, a Microsoft Excel spreadsheet). Once an object is created, you refer to it in code using the object variable you defined. In the following example, you access properties and methods of the new object using the object variable, ExcelSheet, and other Excel objects, including the Application object and the Cells collection. For example:

```
' Make Excel visible through the Application object.
ExcelSheet.Application.Visible = True
' Place some text in the first cell of the sheet.
ExcelSheet.Cells(1,1).Value = "This is column A, row 1"
' Save the sheet.
ExcelSheet.SaveAs "C:\DOCS\TEST.XLS"
' Close Excel with the Quit method on the Application object.
ExcelSheet.Application.Quit
' Release the object variable.
Set ExcelSheet = Nothing
```

CreateTextFile Method

Description

Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax

object.**CreateTextFile(***filename***[, ***overwrite***[, ***unicode***]])**

The **CreateTextFile** method has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject** or **Folder** object.
filename	Required. String expression that identifies the file to create.
overwrite	Optional. Boolean value that indicates if an existing file can be overwritten. The value is **True** if the file can be overwritten; **False** if it can't be overwritten. If

	omitted, existing files are not overwritten.
unicode	Optional. Boolean value that indicates whether the file is created as a Unicode or ASCII file. The value is **True** if the file is created as a Unicode file; **False** if it's created as an ASCII file. If omitted, an ASCII file is assumed.

Remarks

The following code illustrates how to use the **CreateTextFile** method to create and open a text file:

```
Sub CreateAfile
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set a = fs.CreateTextFile("c:\testfile.txt", True)
  a.WriteLine("This is a test.")
  a.Close
End Sub
```

If the *overwrite* argument is **False**, or is not provided, for a *filename* that already exists, an error occurs.

CSng Function

Description

Returns an expression that has been converted to a **Variant** of subtype **Single**.

Syntax

CSng(*expression***)**

The *expression* argument is any valid expression.

Remarks

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CDbl** or **CSng** to force double-precision or single-precision arithmetic in cases where currency or integer arithmetic normally would occur.

Use the **CSng** function to provide internationally aware conversions from any other data type to a **Single** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system, as are different thousand separators.

If *expression* lies outside the acceptable range for the **Single** subtype, an error occurs.

CStr Function

Description

Returns an expression that has been converted to a **Variant** of subtype **String**.

Syntax

CStr(*expression***)**

The *expression* argument is any valid expression.

Remarks

In general, you can document your code using the data type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CStr** to force the result to be expressed as a **String**.

You should use the **CStr** function instead of **Str** to provide internationally aware conversions from any other data type to a **String** subtype. For example, different decimal separators are properly recognized depending on the locale setting of your system.

The data in *expression* determines what is returned according to the following table:

If *expression* is	CStr returns
Boolean	A **String** containing **True** or **False**.
Date	A **String** containing a date in the short-date format of your system.
Null	A run-time error.
Empty	A zero-length **String** ("").
Error	A **String** containing the word Error followed by the error number.

Other numeric	A **String** containing the number.

Date Function

Description

Returns the current system date.

Syntax

Date

DateAddFunction

Description

Returns a date to which a specified time interval has been added.

Syntax

DateAdd(*interval***,** *number***,** *date***)**

The **DateAdd** function syntax has these parts:

Part	Description
interval	Required. String expression that is the interval you want to add. See Settings section for values.
number	Required. Numeric expression that is the number of interval you want to add. The numeric expression can either be positive, for dates in the future, or negative, for dates in the past.
date	Required. **Variant** or literal representing the date to which *interval* is added.

Settings

The *interval* argument can have the following values:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
n	Minute
s	Second

Remarks

You can use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 30 days from today or a time 45 minutes from now. To add days to *date*, you can use Day of Year ("y"), Day ("d"), or Weekday ("w").

The **DateAdd** function won't return an invalid date. The following example adds one month to January 31:

```
NewDate = DateAdd("m", 1, "31-Jan-95")
```

In this case, **DateAdd** returns 28-Feb-95, not 31-Feb-95. If *date* is 31-Jan-96, it returns 29-Feb-96 because 1996 is a leap year.

If the calculated date would precede the year 100, an error occurs.

If number isn't a **Long** value, it is rounded to the nearest whole number before being evaluated.

DateCreated Property

Description

Returns the date and time that the specified file or folder was created. Read-only.

Syntax

object.**DateCreated**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateCreated** property with a file:

```
Sub ShowFileInfo(filespec)
  Dim fs, f, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.GetFile(filespec)
  s = "Created: " & f.DateCreated
  MsgBox s
End Sub
```

DateDiff Function

Description

Returns the number of intervals between two dates.

Syntax

DateDiff(*interval***,** *date1***,** *date2* **[,***firstdayofweek***[,** *firstweekofyear***]])**

The **DateDiff** function syntax has these parts:

Part	Description
interval	Required. String expression that is the interval you want to use to calculate the differences between *date1* and *date2*. See Settings section for values.
date1, date2	Required. Date expressions. Two dates you want to use in the calculation.
firstdayofweek	Optional. Constant that specifies the day of the week. If not specified, Sunday is assumed. See Settings section for values.
firstweekofyear	Optional. Constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. See Settings section for values.

Settings

The *interval* argument can have the following values:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
n	Minute
s	Second

The *firstdayofweek* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The *firstweekofyear* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbFirstJan1	1	Start with the week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the week that has at least four days in the new year.
vbFirstFullWeek	3	Start with the first full weekof the new year.

Remarks

You can use the **DateDiff** function to determine how many specified time intervals exist between two dates. For example, you might use **DateDiff** to calculate the number of days between two dates, or the number of weeks between today and the end of the year.

To calculate the number of days between *date1* and *date2*, you can use either Day of year ("y") or Day ("d"). When *interval* is Weekday ("w"), **DateDiff** returns the number of weeks between the two dates. If *date1* falls on a Monday, **DateDiff** counts the number of Mondays until *date2*. It counts *date2* but not *date1*. If *interval* is Week ("ww"), however, the **DateDiff** function returns the number of calendar weeks between the two dates. It counts the number of Sundays between *date1* and *date2*. **DateDiff** counts *date2* if it falls on a Sunday; but it doesn't count *date1*, even if it does fall on a Sunday.

If *date1* refers to a later point in time than *date2*, the **DateDiff** function returns a negative number.

The *firstdayofweek* argument affects calculations that use the "w" and "ww" interval symbols.

If *date1* or *date2* is a date literal, the specified year becomes a permanent part of that date. However, if *date1* or *date2* is enclosed in quotation marks (" ") and you omit the year, the current year is inserted in your code each time the *date1* or *date2* expression is evaluated. This makes it possible to write code that can be used in different years.

When comparing December 31 to January 1 of the immediately succeeding year, **DateDiff** for Year ("yyyy") returns 1 even though only a day has elapsed.

DateLastAccessed Property

Description

Returns the date and time that the specified file or folder was last accessed. Read-only.

Syntax

object.**DateLastAccessed**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateLastAccessed** property with a file:

```
Sub ShowFileAccessInfo(filespec)
  Dim fs, f, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.GetFile(filespec)
  s = UCase(filespec) & vbCrLf
  s = s & "Created: " & f.DateCreated & vbCrLf
  s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
  s = s & "Last Modified: " & f.DateLastModified
  MsgBox s, 0, "File Access Info"
End Sub
```

Important This method depends on the underlying operating system for its behavior. If the operating system does not support providing time information, none will be returned.

DateLastModified Property

Description

Returns the date and time that the specified file or folder was last modified. Read-only.

Syntax

object.**DateLastModified**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **DateLastModified** property with a file:

```
Sub ShowFileAccessInfo(filespec)
  Dim fs, f, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.GetFile(filespec)
  s = UCase(filespec) & vbCrLf
  s = s & "Created: " & f.DateCreated & vbCrLf
  s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
  s = s & "Last Modified: " & f.DateLastModified
  MsgBox s, 0, "File Access Info"
End Sub
```

DatePart Function

Description

Returns the specified part of a given date.

Syntax

DatePart(*interval*, *date*[, *firstdayofweek*[, *firstweekofyear*]])

The **DatePart** function syntax has these parts:

Part	Description
interval	Required. String expression that is the interval of time you want to return. See Settings section for values.
date	Required. Date expression you want to evaluate.
firstdayof week	Optional. Constant that specifies the day of the week. If not specified, Sunday is assumed. See Settings section for values.
firstweekofyear	Optional. Constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. See Settings section for values.

Settings

The *interval* argument can have the following values:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
n	Minute

s	Second

The *firstdayofweek* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The *firstweekofyear* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbFirstJan1	1	Start with the week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the week that has at least four days in the new year.
vbFirstFullWeek	3	Start with the first full weekof the new year.

Remarks

You can use the **DatePart** function to evaluate a date and return a specific interval of time. For example, you might use **DatePart** to calculate the day of the week or the current hour.

The *firstdayofweek* argument affects calculations that use the "w" and "ww" interval symbols.

If *date* is a date literal, the specified year becomes a permanent part of that date. However, if *date* is enclosed in quotation marks (" "), and you omit the year, the current year is inserted in your code each time the *date* expression is evaluated. This makes it possible to write code that can be used in different years.

DateSerial Function

Description

Returns a **Variant** of subtype **Date** for a specified year, month, and day.

Syntax

DateSerial(*year*, *month*, *day*)

The **DateSerial** function syntax has these arguments:

Part	Description
year	Number between 100 and 9999, inclusive, or a numeric expression.
month	Any numeric expression.
day	Any numeric expression.

Remarks

To specify a date, such as December 31, 1991, the range of numbers for each **DateSerial** argument should be in the accepted range for the unit; that is, 1–31 for days and 1–12 for months. However, you can also specify relative dates for each argument using any numeric expression that represents some number of days, months, or years before or after a certain date.

The following example uses numeric expressions instead of absolute date numbers. Here the **DateSerial** function returns a date that is the day before the first day (1 – 1) of two months before August (8 – 2) of 10 years before 1990 (1990 – 10); in other words, May 31, 1980.

```
DateSerial(1990 - 10, 8 - 2, 1 - 1)
```


For the *year* argument, values between 0 and 99, inclusive, are interpreted as the years 1900–1999. For all other *year* arguments, use a complete four-digit year (for example, 1800).

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 35 days, it is evaluated as one month and some number of days, depending on where in the year it is applied. However, if any single argument is outside the range -32,768 to 32,767, or if the date specified by the three arguments, either directly or by expression, falls outside the acceptable range of dates, an error occurs.

DateValue Function

Description

Returns a **Variant** of subtype **Date**.

Syntax

DateValue(*date*)

The *date* argument is normally a string expression representing a date from January 1, 100 through December 31, 9999. However, *date* can also be any expression that can represent a date, a time, or both a date and time, in that range.

Remarks

If the *date* argument includes time information, **DateValue** doesn't return it. However, if *date* includes invalid time information (such as "89:98"), an error occurs.

If *date* is a string that includes only numbers separated by valid date separators, **DateValue** recognizes the order for month, day, and year according to the short date format you specified for your system. **DateValue** also recognizes unambiguous dates that contain month names, either in long or abbreviated form. For example, in addition to recognizing 12/30/1991 and 12/30/91, **DateValue** also recognizes December 30, 1991 and Dec 30, 1991.

If the year part of *date* is omitted, **DateValue** uses the current year from your computer's system date.

Day Function

Description

Returns a whole number between 1 and 31, inclusive, representing the day of the month.

Syntax

Day(*date*)

The *date* argument is any expression that can represent a date. If *date* contains **Null**, **Null** is returned.

Delete Method

Description

Deletes a specified file or folder.

Syntax

object.**Delete** *force*

The **Delete** method syntax has these parts:

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
force	Optional. Boolean value that is **True** if files or folders with the read-only attribute set are to be deleted; **False** (default) if they are not.

Remarks

An error occurs if the specified file or folder does not exist.

The results of the **Delete** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.DeleteFile** or **FileSystemObject.DeleteFolder**.

The **Delete** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

DeleteFile Method

Description

Deletes a specified file.

Syntax

object.**DeleteFile** *filespec*[, *force*]

The **DeleteFile** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
filespec	Required. The name of the file to delete. The *filespec* can contain wildcard characters in the last path component.
force	Optional. Boolean value that is **True** if files with the read-only attribute set are to be deleted; **False** (default) if they are not.

Remarks

An error occurs if no matching files are found. The **DeleteFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

DeleteFolder Method

Description

Deletes a specified folder and its contents.

Syntax

object.**DeleteFolder** *folderspec*[, *force*]

The **DeleteFolder** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
folderspec	Required. The name of the folder to delete. The *folderspec* can contain wildcard characters in the last path component.
force	Optional. Boolean value that is **True** if folders with the read-only attribute set are to be deleted; **False** (default) if they are not.

Remarks

The **DeleteFolder** method does not distinguish between folders that have contents and those that do not. The specified folder is deleted regardless of whether or not it has contents.

An error occurs if no matching folders are found. The **DeleteFolder** method stops on the first error it encounters. No attempt is made to roll back or undo any changes that were made before an error occurred.

Description Property

Description

Returns or sets a descriptive string associated with an error.

Syntax

object.**Description** [= *stringexpression*]

The **Description** property syntax has these parts:

Part	Description
object	Always the **Err** object.
stringexpression	A string expression containing a description of the error.

Remarks

The **Description** property consists of a short description of the error. Use this property to alert the user to an error that you can't or don't want to handle. When generating a user-defined error, assign a short description of your error to this property. If **Description** isn't filled in, and the value of **Number** corresponds to a VBScript run-time error, the descriptive string associated with the error is returned.

Dictionary Object

Description

Object that stores data key, item pairs.

Syntax

Scripting.Dictionary

Remarks

A **Dictionary** object is the equivalent of a PERL associative array. Items, which can be any form of data, are stored in the array. Each item is associated with a unique key. The key is used to retrieve an individual item and is usually a integer or a string, but can be anything except an array.
The following code illustrates how to create a **Dictionary** object:

```
Dim d                'Create a variable
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"    'Add some keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
```

Dim Statement

Description

Declares variables and allocates storage space.

Syntax

Dim *varname*[([*subscripts*])][**,** *varname*[([*subscripts*])]] **. . .**
The **Dim** statement syntax has these parts:

Part	Description
varname	Name of the variable; follows standard variable naming conventions.
subscripts	Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: *upperbound* [**,***upperbound*] **. . .** The lower bound of an array is always zero.

Remarks

Variables declared with **Dim** at the script level are available to all procedures within the script. At the procedure level, variables are available only within the procedure.
You can also use the **Dim** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Dim** statement, an error occurs.
When variables are initialized, a numeric variable is initialized to 0 and a string is initialized to a zero-length string ("").

Tip When you use the **Dim** statement in a procedure, you generally put the **Dim** statement at the beginning of the procedure.

Division Operator (/)

Description

Used to divide two numbers and return a floating-point result.

Syntax

result **=** *number1*/*number2*
The / operator syntax has these parts:

Part	Description
result	Any numeric variable.
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

If one or both expressions are **Null** expressions, *result* is **Null**. Any expression that is **Empty** is treated as 0.

Do...Loop Statement

Description

Repeats a block of statements while a condition is **True** or until a condition becomes **True**.

Syntax

Do [{**While** | **Until**} *condition*]
[*statements*]
[**Exit Do**]
[*statements*]
Loop
Or, you can use this syntax:
Do
[*statements*]
[**Exit Do**]
[*statements*]
Loop [{**While** | **Until**} *condition*]
The **Do...Loop** statement syntax has these parts:

Part	Description
condition	Numeric or string expression that is **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**.
statements	One or more statements that are repeated while or until *condition* is **True**.

Remarks

The **Exit Do** can only be used within a **Do...Loop** control structure to provide an alternate way to exit a **Do...Loop**. Any number of **Exit Do** statements may be placed anywhere in the **Do...Loop**. Often used with the evaluation of some condition (for example, **If...Then**), **Exit Do** transfers control to the statement immediately following the **Loop**.
When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where it occurs.

Drive Object

Description

Provides access to the properties of a particular disk drive or network share.

Remarks

The following code illustrates the use of the **Drive** object to access drive properties:

```
Sub ShowFreeSpace(drvPath)
  Dim fs, d, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set d = fs.GetDrive(fs.GetDriveName(drvPath))
  s = "Drive " & UCase(drvPath) & " - "
  s = s & d.VolumeName  & vbCrLf
  s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
  s = s & " Kbytes"
  MsgBox s
End Sub
```

Drive Property

Description

Returns the drive letter of the drive on which the specified file or folder resides. Read-only.

Syntax

object.**Drive**
The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Drive** property:

```
Sub ShowFileAccessInfo(filespec)
  Dim fs, f, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.GetFile(filespec)
  s = f.Name & " on Drive " & UCase(f.Drive) & vbCrLf
  s = s & "Created: " & f.DateCreated & vbCrLf
  s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
```



```
  s = s & "Last Modified: " & f.DateLastModified
  MsgBox s, 0, "File Access Info"
End Sub
```

DriveExists Method

Description

Returns **True** if the specified drive exists; **False** if it does not.

Syntax

object.**DriveExists**(*drivespec*)
The **DriveExists** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
drivespec	Required. A drive letter or a complete path specification.

Remarks

For drives with removable media, the **DriveExists** method returns **True** even if there are no media present. Use the **IsReady** property of the **Drive** object to determine if a drive is ready.

DriveLetter Property

Description

Returns the drive letter of a physical local drive or a network share. Read-only.

Syntax

object.**DriveLetter**
The *object* is always a **Drive** object.

Remarks

The **DriveLetter** property returns a zero-length string ("") if the specified drive is not associated with a drive letter, for example, a network share that has not been mapped to a drive letter.
The following code illustrates the use of the **DriveLetter** property:

```
Sub ShowDriveLetter(drvPath)
  Dim fs, d, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set d = fs.GetDrive(fs.GetDriveName(drvPath))
  s = "Drive " & d.DriveLetter & ": - "
  s = s & d.VolumeName  & vbCrLf
  s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
  s = s & " Kbytes"
  MsgBox s
End Sub
```

Drives Collection

Description

Read-only collection of all available drives.

Remarks

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.
The following code illustrates how to get the **Drives** collection and iterate the collection using the **For Each...Next** statement:

```
Sub ShowDriveList
  Dim fs, d, dc, s, n
  Set fs = CreateObject("Scripting.FileSystemObject")
```



```
  Set dc = fs.Drives
  For Each d in dc
    s = s & d.DriveLetter & " - "
    If d.DriveType = Remote Then
      n = d.ShareName
    Else
      n = d.VolumeName
    End If
    s = s & n & vbCrLf
  Next
  MsgBox s
End Sub
```

Drives Property

Description

Returns a **Drives** collection consisting of all **Drive** objects available on the local machine.

Syntax

object.**Drives**
The *object* is always a **FileSystemObject**.

Remarks

Removable-media drives need not have media inserted for them to appear in the **Drives** collection.
You can iterate the members of the **Drives** collection using a **For Each...Next** construct as illustrated in the following code:

```
Sub ShowDriveList
  Dim fs, d, dc, s, n
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set dc = fs.Drives
  For Each d in dc
    s = s & d.DriveLetter & " - "
    If d.DriveType = 3 Then
      n = d.ShareName
    Else
      n = d.VolumeName
    End If
    s = s & n & vbCrLf
  Next
  MsgBox s
End Sub
```

DriveType Property

Description

Returns a value indicating the type of a specified drive.

Syntax

object.**DriveType**
The *object* is always a **Drive** object.

Remarks

The following code illustrates the use of the **DriveType** property:

```
Sub ShowDriveType(drvpath)
  Dim fs, d, s, t
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set d = fs.GetDrive(drvpath)
  Select Case d.DriveType
    Case 0: t = "Unknown"
    Case 1: t = "Removable"
    Case 2: t = "Fixed"
    Case 3: t = "Network"
    Case 4: t = "CD-ROM"
    Case 5: t = "RAM Disk"
  End Select
  s = "Drive " & d.DriveLetter & ": - " & t
  MsgBox s
End Sub
```

Empty

Description

The **Empty** keyword is used to indicate an uninitialized variable value. This is not the same thing as **Null**.

Eqv Operator

Description

Used to perform a logical equivalence on two expressions.

Syntax

result **=** *expression1* **Eqv** *expression2*

The **Eqv** operator syntax has these parts:

Part	Description
result	Any numeric variable.
expression1	Any expression.
expression2	Any expression.

Remarks

If either expression is **Null**, *result* is also **Null**. When neither expression is **Null**, *result* is determined according to the following table:

If *expression1* is	And *expression2* is	The *result* is
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**True**

The **Eqv** operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	The *result* is
0	0	1
0	1	0
1	0	0
1	1	1

Erase Statement

Description

Reinitializes the elements of fixed-size arrays and deallocates dynamic-array storage space.

Syntax

Erase *array*

The *array* argument is the name of the array variable to be erased.

Remarks

It is important to know whether an array is fixed-size (ordinary) or dynamic because **Erase** behaves differently depending on the type of array. **Erase** recovers no memory for fixed-size arrays. **Erase** sets the elements of a fixed array as follows:

Type of array	Effect of Erase on fixed-array elements
Fixed numeric array	Sets each element to zero.
Fixed string array	Sets each element to zero-length ("").
Array of objects	Sets each element to the special value Nothing.

Erase frees the memory used by dynamic arrays. Before your program can refer to the dynamic array again, it must redeclare the array variable's dimensions using a **ReDim** statement.

Err Object

Description

Contains information about run-time errors. Accepts the **Raise** and **Clear** methods for generating and clearing run-time errors.

Syntax

Err[**.**{*property* | *method*}]

Remarks

The properties of the **Err** object are set by the generator of an error — Visual Basic, an Automation object, or the VBScript programmer.

The default property of the **Err** object is **Number**. **Err.Number** contains an integer and can be used by an Automation object to return an SCODE.

When a run-time error occurs, the properties of the **Err** object are filled with information that uniquely identifies the error and information that can be used to handle it. To generate a run-time error in your code, use the **Raise** method.

The **Err** object's properties are reset to zero or zero-length strings ("") after an **On Error Resume Next** statement. The **Clear** method can be used to explicitly reset **Err**.

The **Err** object is an intrinsic object with global scope — there is no need to create an instance of it in your code.

Exists Method

Description

Returns **True** if a specified key exists in the **Dictionary** object, **False** if it does not.

Syntax

object.**Exists(***key***)**

The **Exists** method syntax has these parts:

Part	Description
object	Required. Always the name of a **Dictionary** object.
key	Required. *Key* value being searched for in the **Dictionary** object.

Exit Statement

Description

Exits a block of **Do...Loop**, **For...Next**, **Function**, or **Sub** code.

Syntax

Exit Do
Exit For
Exit Function
Exit Sub

The **Exit** statement syntax has these forms:

Statement	Description
Exit Do	Provides a way to exit a **Do...Loop** statement. It can be used only inside a **Do...Loop** statement. **Exit Do** transfers control to the statement following the **Loop** statement. When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where it occurs.
Exit For	Provides a way to exit a **For** loop. It can be used only in a **For...Next** or **For Each...Next** loop. **Exit For** transfers control to the statement following the **Next** statement. When used within nested **For** loops, **Exit For** transfers control to the loop that is one nested level above the loop where it occurs.
Exit Function	Immediately exits the **Function** procedure in which it appears. Execution continues with the statement following the statement that called the **Function**.
Exit Sub	Immediately exits the **Sub** procedure in which it appears. Execution continues with the statement following the statement that called the **Sub**.

Exp Function

Description

Returns *e* (the base of natural logarithms) raised to a power.

Syntax

Exp(*number***)**

The *number* argument can be any valid numeric expression.

Remarks

If the value of *number* exceeds 709.782712893, an error occurs. The constant *e* is approximately 2.718282.

Note The **Exp** function complements the action of the **Log** function and is sometimes referred to as the antilogarithm.

Exponentiation Operator (^)

Description

Used to raise a number to the power of an exponent.

Syntax

result **=** *number***^***exponent*

The **^** operator syntax has these parts:

Part	Description
result	Any numeric variable.
number	Any numeric expression.
exponent	Any numeric expression.

Remarks

Number can be negative only if *exponent* is an integer value. When more than one exponentiation is performed in a single expression, the **^** operator is evaluated as it is encountered from left to right.

If either *number* or *exponent* is a **Null** expression, *result* is also **Null**.

False

Description

The **False** keyword has a value equal to 0.

FileExists Method

Description

Returns **True** if a specified file exists; **False** if it does not.

Syntax

object.**FileExists(***filespec***)**

The **FileExists** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
filespec	Required. The name of the file whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the file isn't expected to exist in the current folder.

File Object

Description

Provides access to all the properties of a file.

Remarks

The following code illustrates how to obtain a **File** object and how to view one of its properties.

```
Sub ShowFileInfo(filespec)
   Dim fs, f, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFile(filespec)
   s = f.DateCreated
   MsgBox s
End Sub
```

Files Collection

Description

Collection of all **File** objects within a folder.

Remarks

The following code illustrates how to get a **Files** collection and iterate the collection using the **For Each...Next** statement:

```
Sub ShowFolderList(folderspec)
   Dim fs, f, f1, fc, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFolder(folderspec)
   Set fc = f.Files
   For Each f1 in fc
      s = s & f1.name
      s = s & vbCrLf
   Next
   MsgBox s
End Sub
```

Files Property

Description

Returns a **Files** collection consisting of all **File** objects contained in the specified folder, including those with hidden and system file attributes set.

Syntax

object.**Files**

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **Files** property:

```
Sub ShowFileList(folderspec)
   Dim fs, f, f1, fc, s
```

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set f = fs.GetFolder(folderspec)
Set fc = f.Files
For Each f1 in fc
   s = s & f1.name
   s = s &  vbCrLf
Next
MsgBox s
End Sub
```

FileSystemObject Object

Description

Provides access to a computer's file system.

Syntax

Scripting.FileSystemObject

Remarks

The following code illustrates how the **FileSystemObject** is used to return a **TextStream** object that can be read from or written to:

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("c:\testfile.txt", True)
a.WriteLine("This is a test.")
a.Close
```

In the code shown above, the **CreateObject** function returns the **FileSystemObject** (fs). The **CreateTextFile** method then creates the file as a **TextStream** object (a) and the **WriteLine** method writes a line of text to the created text file. The **Close** method flushes the buffer and closes the file.

FileSystem Property

Description

Returns the type of file system in use for the specified drive.

Syntax

object.**FileSystem**

The *object* is always a **Drive** object.

Remarks

Available return types include FAT, NTFS, and CDFS.

The following code illustrates the use of the **FileSystem** property:

```
Sub ShowFileSystemType
   Dim fs,d, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive("e:")
   s = d.FileSystem
   MsgBox s
End Sub
```

Filter Function

Description

Returns a zero-based array containing subset of a string array based on a specified filter criteria.

Syntax

Filter(*InputStrings***,** *Value***[,** *Include***[,** *Compare***]])**

The **Filter** function syntax has these parts:

Part	Description
InputStrings	Required. One-dimensional array of strings to be searched.
Value	Required. String to search for.

Include	Optional. Boolean value indicating whether to return substrings that include or exclude *Value*. If *Include* is **True**, **Filter** returns the subset of the array that contains *Value* as a substring. If *Include* is **False**, **Filter** returns the subset of the array that does not contain *Value* as a substring.
Compare	Optional. Numeric value indicating the kind of string comparison to use. See Settings section for values.

Settings

The *Compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Remarks

If no matches of *Value* are found within *InputStrings*, **Filter** returns an empty array. An error occurs if *InputStrings* is **Null** or is not a one-dimensional array.

The array returned by the **Filter** function contains only enough elements to contain the number of matched items.

Fix Function

Description

Returns the integer portion of a number.

Syntax

Int(*number***)**

Fix(*number***)**

The *number* argument can be any valid numeric expression. If *number* contains **Null**, **Null** is returned.

Remarks

Both **Int** and **Fix** remove the fractional part of *number* and return the resulting integer value.

The difference between **Int** and **Fix** is that if *number* is negative, **Int** returns the first negative integer less than or equal to *number*, whereas **Fix** returns the first negative integer greater than or equal to *number*. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Fix(*number***)** is equivalent to:

```
Sgn(number) * Int(Abs(number)Int)
```

Folder Object

Description

Provides access to all the properties of a folder.

Remarks

The following code illustrates how to obtain a **Folder** object and how to return one of its properties:

```
Sub ShowFolderInfo(folderspec)
   Dim fs, f, s,
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFolder(folderspec)
   s = f.DateCreated
   MsgBox s
End Sub
```


Folders Collection

Description

Collection of all **Folder** objects contained within a **Folder** object.

Remarks

The following code illustrates how to get a **Folders** collection and how to iterate the collection using the **For Each...Next** statement:

```
Sub ShowFolderList(folderspec)
   Dim fs, f, f1, fc, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFolder(folderspec)
   Set fc = f.SubFolders
   For Each f1 in fc
      s = s & f1.name
      s = s &  vbCrLf
   Next
   MsgBox s
End Sub
```

FolderExists Method

Description

Returns **True** if a specified folder exists; **False** if it does not.

Syntax

object.**FolderExists(***folderspec***)**

The **FolderExists** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
folderspec	Required. The name of the folder whose existence is to be determined. A complete path specification (either absolute or relative) must be provided if the folder isn't expected to exist in the current folder.

For...Next Statement

Description

Repeats a group of statements a specified number of times.

Syntax

For *counter* **=** *start* **To** *end* [**Step** *step*]
   [*statements*]
   [**Exit For**]
   [*statements*]
Next

The **For...Next** statement syntax has these parts:

Part	Description
counter	Numeric variable used as a loop counter. The variable can't be an array element or an element of a user-defined type.
start	Initial value of *counter*.
end	Final value of *counter*.
step	Amount *counter* is changed each time through the loop. If not specified, *step* defaults to one.
statements	One or more statements between **For** and **Next** that are executed the specified number of times.

Remarks

The *step* argument can be either positive or negative. The value of the *step* argument determines loop processing as follows:

Value	Loop executes if
Positive or 0	*counter* <= *end*
Negative	*counter* >= *end*

Once the loop starts and all statements in the loop have executed, *step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the **Next** statement.

Tip Changing the value of *counter* while inside a loop can make it more difficult to read and debug your code.

Exit For can only be used within a **For Each...Next** or **For...Next** control structure to provide an alternate way to exit. Any number of **Exit For** statements may be placed anywhere in the loop. **Exit For** is often used with the evaluation of some condition (for example, **If...Then**), and transfers control to the statement immediately following **Next**.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its *counter*. The following construction is correct:

```
For I = 1 To 10
   For J = 1 To 10
      For K = 1 To 10
      . . .
      Next
   Next
Next
```

For Each...Next Statement

Description

Repeats a group of statements for each element in an array or collection.

Syntax

For Each *element* **In** *group*
   [*statements*]
   [**Exit For**]
   [*statements*]
Next [*element*]

The **For Each...Next** statement syntax has these parts:

Part	Description
element	Variable used to iterate through the elements of the collection or array. For collections, *element* can only be a **Variant** variable, a generic **Object** variable, or any specific Automation object variable. For arrays, *element* can only be a **Variant** variable.
group	Name of an object collection or array.
statements	One or more statements that are executed on each item in *group*.

Remarks

The **For Each** block is entered if there is at least one element in *group*. Once the loop has been entered, all the statements in the loop are executed for the first element in *group*. As long as there are more elements in *group*, the statements in the loop continue to execute for each element. When there are no more elements in *group*, the loop is exited and execution continues with the statement following the **Next** statement.

The **Exit For** can only be used within a **For Each...Next** or **For...Next** control structure to provide an alternate way to exit. Any number of **Exit For** statements may be placed anywhere in the loop. The **Exit For** is often used with the evaluation of some condition (for example, **If...Then**), and transfers control to the statement immediately following **Next**.
You can nest **For Each...Next** loops by placing one **For Each...Next** loop within another. However, each loop *element* must be unique.

Note If you omit *element* in a **Next** statement, execution continues as if you had included it. If a **Next** statement is encountered before its corresponding **For** statement, an error occurs.

FormatCurrency Function

Description

Returns an expression formatted as a currency value using the currency symbol defined in the system control panel.

Syntax

FormatCurrency(*Expression*[**,***NumDigitsAfterDecimal* [**,***IncludeLeadingDigit* [**,***UseParensForNegativeNumbers* [**,***GroupDigits*]]]])

The **FormatCurrency** function syntax has these parts:

Part	Description
Expression	Required. Expression to be formatted.
NumDigitsAfterDecimal	Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.
IncludeLeadingDigit	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values.
UseParensForNegativeNumbers	Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values.
GroupDigits	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings. See Settings section for values.

Settings

The *IncludeLeadingDigit*, *UseParensForNegativeNumbers*, and *GroupDigits* arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Remarks

When one or more optional arguments are omitted, values for omitted arguments are provided by the computer's regional settings.
The position of the currency symbol relative to the currency value is determined by the system's regional settings.
Note All settings information comes from the Regional Settings Currency tab, except leading zero which comes from the Number tab.

FormatDateTime Function

Description

Returns an expression formatted as a date or time.

Syntax

FormatDateTime(*Date*[**,***NamedFormat*])

The **FormatDateTime** function syntax has these parts:

Part	Description
Date	Required. Date expression to be formatted.
NamedFormat	Optional. Numeric value that indicates the date/time format used. If omitted, **vbGeneralDate** is used.

Settings

The *NamedFormat* argument has the following settings:

Constant	Value	Description
vbGeneralDate	0	Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.
vbLongDate	1	Display a date using the long date format specified in your computer's regional settings.
vbShortDate	2	Display a date using the short date format specified in your computer's regional settings.
vbLongTime	3	Display a time using the time format specified in your computer's regional settings.
vbShortTime	4	Display a time using the 24-hour format (hh:mm).

FormatNumber Function

Description

Returns an expression formatted as a number.

Syntax

FormatNumber(*Expression*[**,***NumDigitsAfterDecimal* [**,***IncludeLeadingDigit* [**,***UseParensForNegativeNumbers* [**,***GroupDigits*]]]])

The **FormatNumber** function syntax has these parts:

Part	Description
Expression	Required. Expression to be formatted.
NumDigitsAfterDecimal	Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.
IncludeLeadingDigit	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values.
UseParensForNegativeNumbers	Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values.
GroupDigits	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in

	the control panel. See Settings section for values.

Settings

The *IncludeLeadingDigit*, *UseParensForNegativeNumbers*, and *GroupDigits* arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Remarks

When one or more of the optional arguments are omitted, the values for omitted arguments are provided by the computer's regional settings.

Note All settings information comes from the Regional Settings Number tab.

FormatPercent Function

Description

Returns an expression formatted as a percentage (multiplied by 100) with a trailing % character.

Syntax

FormatPercent(*Expression*[**,***NumDigitsAfterDecimal* [**,***IncludeLeadingDigit* [**,***UseParensForNegativeNumbers* [**,***GroupDigits*]]]])

The **FormatPercent** function syntax has these parts:

Part	Description
Expression	Required. Expression to be formatted.
NumDigitsAfterDecimal	Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.
IncludeLeadingDigit	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. See Settings section for values.
UseParensForNegativeNumbers	Optional. Tristate constant that indicates whether or not to place negative values within parentheses. See Settings section for values.
GroupDigits	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the control panel. See Settings section for values.

Settings

The *IncludeLeadingDigit*, *UseParensForNegativeNumbers*, and *GroupDigits* arguments have the following settings:

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use the setting from the computer's regional settings.

Remarks

When one or more optional arguments are omitted, the values for the omitted arguments are provided by the computer's regional settings.

Note All settings information comes from the Regional Settings Number tab.

FreeSpace Property

Description

Returns the amount of free space available to a user on the specified drive or network share. Read-only.

Syntax

object.**FreeSpace**

The *object* is always a **Drive** object.

Remarks

The value returned by the **FreeSpace** property is typically the same as that returned by the **AvailableSpace** property. Differences may occur between the two for computer systems that support quotas.

The following code illustrates the use of the **FreeSpace** property:

```
Sub ShowFreeSpace(drvPath)
   Dim fs, d, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(fs.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & d.VolumeName  & vbCrLf
   s = s & "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)
   s = s & " Kbytes"
   MsgBox s
End Sub
```

Function Statement

Description

Returns a reference to an Automation object from a file.

Syntax

GetObject([*pathname*] [**,** *class*])

The **GetObject** function syntax has these parts:

Part	Description
pathname	Optional; String. Full path and name of the file containing the object to retrieve. If *pathname* is omitted, *class* is required.
class	Optional; String. Class of the object.

The *class* argument uses the syntax *appname.objectype* and has these parts:

Part	Description
appname	Required; String. Name of the application providing the object.
objectype	Required; String. Type or class of object to create.

Remarks

Use the **GetObject** function to access an Automation object from a file and assign the object to an object variable. Use the **Set** statement to assign the object returned by **GetObject** to the object variable. For example:

```
Dim CADObject
Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```

When this code is executed, the application associated with the specified pathname is started and the object in the specified file is activated. If *pathname* is a zero-length string (""), **GetObject** returns a new object instance of the specified type. If the *pathname* argument is omitted, **GetObject** returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (!) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called SCHEMA.CAD:

```
Set LayerObject = GetObject("C:\CAD\SCHEMA.CAD!Layer3")
```

If you don't specify the object's class, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an Application object, a Drawing object, and a Toolbar object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional *class* argument. For example:

```
Dim MyObject
```

```
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW",
"FIGMENT.DRAWING")
```

In the preceding example, FIGMENT is the name of a drawing application and DRAWING is one of the object types it supports. Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable MyObject. For example:

```
MyObject.Line 9, 90
MyObject.InsertText 9, 100, "Hello, world."
MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

Note Use the **GetObject** function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the **CreateObject** function.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the *pathname* argument is omitted.

GetAbsolutePathName Method

Description

Returns a complete and unambiguous path from a provided path specification.

Syntax

object.**GetAbsolutePathName**(*pathspec*)

The **GetAbsolutePathName** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
pathspec	Required. Path specification to change to a complete and unambiguous path.

Remarks

A path is complete and unambiguous if it provides a complete reference from the root of the specified drive. A complete path can only end with a path separator character (\) if it specifies the root folder of a mapped drive.

Assuming the current directory is c:\mydocuments\reports, the following table illustrates the behavior of the **GetAbsolutePathName** method.

pathspec	Returned path
"c:"	"c:\mydocuments\reports"
"c:.."	"c:\mydocuments"
"c:\\\"	"c:\"
"c:*.*\may97"	"c:\mydocuments\reports*.*\may97"

	returned.

Remarks

The **GetDriveName** method returns a zero-length string ("") if the drive can't be determined.

Note The **GetDriveName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

GetExtensionName Method

Description

Returns a string containing the extension name for the last component in a path.

Syntax

object.**GetExtensionName**(*path*)

The **GetExtensionName** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
path	Required. The path specification for the component whose extension name is to be returned.

Remarks

For network drives, the root directory (\) is considered to be a component.

The **GetExtensionName** method returns a zero-length string ("") if no component matches the *path* argument.

GetFile Method

Description

Returns a **File** object corresponding to the file in a specified path.

Syntax

object.**GetFile**(*filespec*)

The **GetFile** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
filespec	Required. The *filespec* is the path (absolute or relative) to a specific file.

Remarks

An error occurs if the specified file does not exist.

GetFileName Method

Description

Returns the last component of specified path that is not part of the drive specification.

Syntax

object.**GetFileName**(*pathspec*)

The **GetFileName** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
pathspec	Required. The path (absolute or relative) to a specific file.

Remarks

The **GetFileName** method returns a zero-length string ("") if *pathspec* does not end with the named component.

Note The **GetFileName** method works only on the provided path string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

"region1"	"c:\mydocuments\reports\region1"
"c:\..\..\mydocuments"	"c:\mydocuments"

GetBaseName Method

Description

Returns a string containing the base name of the last component, less any file extension, in a path.

Syntax

object.**GetBaseName**(*path*)

The **GetBaseName** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
path	Required. The path specification for the component whose base name is to be returned.

Remarks

The **GetBaseName** method returns a zero-length string ("") if no component matches the *path* argument.

Note The **GetBaseName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

GetDrive Method

Description

Returns a **Drive** object corresponding to the drive in a specified path.

Syntax

object.**GetDrive** *drivespec*

The **GetDrive** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
drivespec	Required. The *drivespec* argument can be a drive letter (c), a drive letter with a colon appended (c:), a drive letter with a colon and path separator appended (c:\), or any network share specification (\\computer2\share1).

Remarks

For network shares, a check is made to ensure that the share exists.

An error occurs if *drivespec* does not conform to one of the accepted forms or does not exist.

To call the **GetDrive** method on a normal path string, use the following sequence to get a string that is suitable for use as *drivespec*:

```
DriveSpec = GetDriveName(GetAbsolutePathName(Path))
```

GetDriveName Method

Description

Returns a string containing the name of the drive for a specified path.

Syntax

object.**GetDriveName**(*path*)

The **GetDriveName** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
path	Required. The path specification for the component whose drive name is to be

GetFolder Method

Description

Returns a **Folder** object corresponding to the folder in a specified path.

Syntax

object.**GetFolder**(*folderspec*)

The **GetFolder** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
folderspec	Required. The *folderspec* is the path (absolute or relative) to a specific folder.

Remarks

An error occurs if the specified folder does not exist.

GetObject Function

Description

Returns a reference to an Automation object from a file.

Syntax

GetObject([*pathname*] [, *class*])

The **GetObject** function syntax has these parts:

Part	Description
pathname	Optional; String. Full path and name of the file containing the object to retrieve. If *pathname* is omitted, *class* is required.
class	Optional; String. Class of the object.

The *class* argument uses the syntax *appname.objectype* and has these parts:

Part	Description
appname	Required; String. Name of the application providing the object.
objectype	Required; String. Type or class of object to create.

Remarks

Use the **GetObject** function to access an Automation object from a file and assign the object to an object variable. Use the **Set** statement to assign the object returned by **GetObject** to the object variable. For example:

```
Dim CADObject
Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```

When this code is executed, the application associated with the specified pathname is started and the object in the specified file is activated. If *pathname* is a zero-length string (""), **GetObject** returns a new object instance of the specified type. If the *pathname* argument is omitted, **GetObject** returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (!) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called SCHEMA.CAD:

```
Set LayerObject =
GetObject("C:\CAD\SCHEMA.CAD!Layer3")
```

If you don't specify the object's class, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an Application object, a Drawing object, and a Toolbar

object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional *class* argument. For example:

```
Dim MyObject
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW",
"FIGMENT.DRAWING")
```

In the preceding example, `FIGMENT` is the name of a drawing application and `DRAWING` is one of the object types it supports. Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable `MyObject`. For example:

```
MyObject.Line 9, 90
MyObject.InsertText 9, 100, "Hello, world."
MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

Note Use the **GetObject** function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the **CreateObject** function.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the *pathname* argument is omitted.

GetParentFolderName Method

Description

Returns a string containing the name of the parent folder of the last component in a specified path.

Syntax

object.**GetParentFolderName(***path***)**

The **GetParentFolderName** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
path	Required. The path specification for the component whose parent folder name is to be returned.

Remarks

The **GetParentFolderName** method returns a zero-length string ("") if there is no parent folder for the component specified in the *path* argument.

Note The **GetParentFolderName** method works only on the provided *path* string. It does not attempt to resolve the path, nor does it check for the existence of the specified path.

GetSpecialFolder Method

Description

Returns the special folder specified.

Syntax

object.**GetSpecialFolder(***folderspec***)**

The **GetSpecialFolder** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
folderspec	Required. The name of the special folder to be returned. Can be any of the constants shown in the Settings section.

Settings

The *folderspec* argument can have any of the following values:

Constant	Value	Description
WindowsFolder	0	The Windows folder contains files installed by the Windows operating system.
SystemFolder	1	The System folder contains libraries, fonts, and device drivers.
TemporaryFolder	2	The Temp folder is used to store temporary files. Its path is found in the TMP environment variable.

GetTempName Method

Description

Returns a randomly generated temporary file or folder name that is useful for performing operations that require a temporary file or folder.

Syntax

object.**GetTempName**

The optional *object* is always the name of a **FileSystemObject**.

Syntax

The **GetTempName** method does not create a file. It provides only a temporary file name that can be used with **CreateTextFile** to create a file.

Hex Function

Description

Returns a string representing the hexadecimal value of a number.

Syntax

Hex(*number***)**

The *number* argument is any valid expression.

Remarks

If *number* is not already a whole number, it is rounded to the nearest whole number before being evaluated.

If *number* is	Hex returns
Null	**Null**.
Empty	Zero (0).
Any other number	Up to eight hexadecimal characters.

You can represent hexadecimal numbers directly by preceding numbers in the proper range with &H. For example, &H10 represents decimal 16 in hexadecimal notation.

HelpContext Property

Description

Sets or returns a fully qualified path to a Help File.

Syntax

object.**HelpFile** [= *contextID*]

The **HelpFile** property syntax has these parts:

Part	Description
object	Required. Always the **Err** object.
contextID	Optional. Fully qualified path to the Help file.

Remarks

If a Help file is specified in **HelpFile**, it is automatically called when the user clicks the Help button (or presses the F1 key) in the error message dialog box. If the **HelpContext** property contains a valid context ID for the specified file, that topic is automatically displayed. If no **HelpFile** is specified, the VBScript Help file is displayed.

HelpFile Property

Description

Sets or returns a fully qualified path to a Help File.

Syntax

object.**HelpFile** [= *contextID*]

The **HelpFile** property syntax has these parts:

Part	Description
object	Required. Always the **Err** object.
contextID	Optional. Fully qualified path to the Help file.

Remarks

If a Help file is specified in **HelpFile**, it is automatically called when the user clicks the Help button (or presses the F1 key) in the error message dialog box. If the **HelpContext** property contains a valid context ID for the specified file, that topic is automatically displayed. If no **HelpFile** is specified, the VBScript Help file is displayed.

Hour Function

Description

Returns a whole number between 0 and 23, inclusive, representing the hour of the day.

Syntax

Hour(*time***)**

The *time* argument is any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

Description

Returns a whole number between 0 and 23, inclusive, representing the hour of the day.

Syntax

Hour(*time***)**

The *time* argument is any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

If...Then...Else Statement

Description

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

If *condition* **Then** *statements* [**Else** *elsestatements*]

Or, you can use the block form syntax:

If *condition* **Then**
  [*statements*]
[**ElseIf** *condition-n* **Then**
  [*elseifstatements*]] . . .
[**Else**
  [*elsestatements*]]
End If

The **If...Then...Else** statement syntax has these parts:

Part	Description
condition	One or more of the following two types of expressions: A numeric or string expression that evaluates to **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**. An expression of the form **TypeOf** *objectname* **Is** *objecttype*. The *objectname* is any object reference and *objecttype* is any valid object type. The expression is **True** if *objectname* is of the object type specified by *objecttype*; otherwise it is **False**.
statements	One or more statements separated by colons; executed if *condition* is **True**.
condition-n	Same as *condition*.
elseifstatements	One or more statements executed if the associated *condition-n* is **True**.
elsestatements	One or more statements executed if no previous *condition* or *condition-n* expression is **True**.

Remarks

You can use the single-line form (first syntax) for short, simple tests. However, the block form (second syntax) provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug.

Note With the single-line syntax, it is possible to have multiple statements executed as the result of an **If...Then** decision, but they must all be on the same line and separated by colons, as in the following statement:

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

When executing a block **If** (second syntax), *condition* is tested. If *condition* is **True**, the statements following **Then** are executed. If *condition* is **False**, each **ElseIf** (if any) is evaluated in turn. When a **True** condition is found, the statements following the associated **Then** are executed. If none of the **ElseIf** statements are **True** (or there are no **ElseIf** clauses), the statements following **Else** are executed. After executing the statements following **Then** or **Else**, execution continues with the statement following **End If**.

The **Else** and **ElseIf** clauses are both optional. You can have as many **ElseIf** statements as you want in a block **If**, but none can appear after the **Else** clause. Block **If** statements can be nested; that is, contained within one another.

What follows the **Then** keyword is examined to determine whether or not a statement is a block **If**. If anything other than a comment appears after **Then** on the same line, the statement is treated as a single-line **If** statement.

A block **If** statement must be the first statement on a line. The block **If** must end with an **End If** statement.

Imp Operator

Description

Used to perform a logical implication on two expressions.

Syntax

result = *expression1* **Imp** *expression2*

The **Imp** operator syntax has these parts:

Part	Description
result	Any numeric variable.
expression1	Any expression.
expression2	Any expression.

Remarks

The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	Then *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**

The **Imp** operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	Then *result* is
0	0	1
0	1	1
1	0	0
1	1	1

InputBox Function

Description

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns the contents of the text box.

Syntax

InputBox(*prompt*[, *title*][, *default*][, *xpos*][, *ypos*][, *helpfile*, *context*])

The **InputBox** function syntax has these arguments:

Part	Description
prompt	String expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used. If *prompt* consists of more than one line, you can separate the lines using a carriage return character (**Chr**(13)), a linefeed character (**Chr**(10)), or carriage return–linefeed character combination (**Chr**(13) **&** **Chr**(10)) between each line.
title	String expression displayed in the title bar of the dialog box. If you omit *title*, the application name is placed in the title bar.
default	String expression displayed in the text box as the default response if no other input is provided. If you omit *default*, the text box is displayed empty.
xpos	Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If *xpos* is omitted, the dialog box is horizontally centered.
ypos	Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If *ypos* is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.
helpfile	String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If *helpfile* is provided, *context* must also be provided.
context	Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If *context* is provided, *helpfile* must also be provided.

Remarks

When both *helpfile* and *context* are supplied, a Help button is automatically added to the dialog box.
If the user clicks **OK** or presses **ENTER**, the **InputBox** function returns whatever is in the text box.
If the user clicks **Cancel**, the function returns a zero-length string ("").

InStr Function

Description

Returns the position of the first occurrence of one string within another.

Syntax

InStr([*start*,]*string1*, *string2*[, *compare*])

The **InStr** function syntax has these arguments:

Part	Description
start	Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If *start* contains **Null**, an error occurs. The *start* argument is required if *compare* is specified.
string1	Required. String expression being searched.
string2	Required. String expression searched for.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values. If omitted, a binary comparison is performed.

Settings

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.

vbTextCompare	1	Perform a textual comparison.

Return Values

The **InStr** function returns the following values:

If	InStr returns
string1 is zero-length	0
string1 is **Null**	**Null**
string2 is zero-length	*start*
string2 is **Null**	**Null**
string2 is not found	0
string2 is found within *string1*	Position at which match is found
start > **Len**(*string2*)	0

Note

The InStrB function is used with byte data contained in a string. Instead of returning the character position of the first occurrence of one string within another, InStrB returns the byte position.

InStrRev Function

Description

Returns the position of an occurrence of one string within another, from the end of string.

Syntax

InStrRev(*string1*, *string2*[, *start*[, *compare*]])

The **InStrRev** function syntax has these parts:

Part	Description
string1	Required. String expression being searched.
string2	Required. String expression being searched for.
start	Optional. Numeric expression that sets the starting position for each search. If omitted, -1 is used, which means that the search begins at the last character position. If *start* contains **Null**, an error occurs.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. If omitted, a binary comparison is performed. See Settings section for values.

Settings

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values

InStrRev returns the following values:

If	InStrRev returns
string1 is zero-length	0
string1 is **Null**	**Null**
string2 is zero-length	*start*
string2 is **Null**	**Null**
string2 is not found	0

string2 is found within *string1*	Position at which match is found
start > **Len**(*string2*)	0

Remarks

Note that the syntax for the **InStrRev** function is not the same as the syntax for the **InStr** function.

Int Function

Description

Returns the integer portion of a number.

Syntax

Int(*number*)
Fix(*number*)

The *number* argument can be any valid numeric expression. If *number* contains **Null**, **Null** is returned.

Remarks

Both **Int** and **Fix** remove the fractional part of *number* and return the resulting integer value.

The difference between **Int** and **Fix** is that if *number* is negative, **Int** returns the first negative integer less than or equal to *number*, whereas **Fix** returns the first negative integer greater than or equal to *number*. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Fix(*number*) is equivalent to:

```
Sgn(number) * Int(Abs(number)Int)
```

Integer Division Operator (\)

Description

Used to divide two numbers and return an integer result.

Syntax

result = *number1**number2*

The \ operator syntax has these parts:

Part	Description
result	Any numeric variable.
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

Before division is performed, numeric expressions are rounded to **Byte**, **Integer**, or **Long** subtype expressions.

If any expression is **Null**, *result* is also **Null**. Any expression that is **Empty** is treated as 0.

Is Operator

Description

Used to compare two object reference variables.

Syntax

result = *object1* **Is** *object2*

The **Is** operator syntax has these parts:

Part	Description
result	Any numeric variable.
object1	Any object name.
object2	Any object name.

Remarks

If *object1* and *object2* both refer to the same object, *result* is **True**; if they do not, *result* is **False**. Two variables can be made to refer to the same object in several ways. In the following example, A has been set to refer to the same object as B:

```
Set A = B
```

The following example makes A and B refer to the same object as C:

```
Set A = C
Set B = C
```

IsArray Function

Description

Returns a Boolean value indicating whether a variable is an array.

Syntax

IsArray(*varname***)**

The *varname* argument can be any variable.

Remarks

IsArray returns **True** if the variable is an array; otherwise, it returns **False**. **IsArray** is especially useful with variants containing arrays.

IsDate Function

Description

Returns a Boolean value indicating whether an expression can be converted to a date.

Syntax

IsDate(*expression***)**

The *expression* argument can be any date expression or string expression recognizable as a date or time.

Remarks

IsDate returns **True** if the expression is a date or can be converted to a valid date; otherwise, it returns **False**. In Microsoft Windows, the range of valid dates is January 1, 100 A.D. through December 31, 9999 A.D.; the ranges vary among operating systems.

IsEmpty Function

Description

Returns a Boolean value indicating whether a variable has been initialized.

Syntax

IsEmpty(*expression***)**

The *expression* argument can be any expression. However, because **IsEmpty** is used to determine if individual variables are initialized, the *expression* argument is most often a single variable name.

Remarks

IsEmpty returns **True** if the variable is uninitialized, or is explicitly set to **Empty**; otherwise, it returns **False**. **False** is always returned if *expression* contains more than one variable.

IsNull Function

Description

Returns a Boolean value that indicates whether an expression contains no valid data (**Null**).

Syntax

IsNull(*expression***)**

The *expression* argument can be any expression.

Remarks

IsNull returns **True** if *expression* is **Null**, that is, it contains no valid data; otherwise, **IsNull** returns **False**. If *expression* consists of more than one variable, **Null** in any constituent variable causes **True** to be returned for the entire expression.

The **Null** value indicates that the variable contains no valid data. **Null** is not the same as **Empty**, which indicates that a variable has not yet been initialized. It is also not the same as a zero-length string (""), which is sometimes referred to as a null string.

Important Use the **IsNull** function to determine whether an expression contains a **Null** value. Expressions that you might expect to evaluate to **True** under some circumstances, such as If Var = Null and If Var <> Null, are always **False**. This is because any expression containing a **Null** is itself **Null**, and therefore, **False**.

IsNumeric Function

Description

Returns a Boolean value indicating whether an expression can be evaluated as a number.

Syntax

IsNumeric(*expression***)**

The *expression* argument can be any expression.

Remarks

IsNumeric returns **True** if the entire *expression* is recognized as a number; otherwise, it returns **False**.

IsNumeric returns **False** if *expression* is a date expression.

IsObject Function

Description

Returns a Boolean value indicating whether an expression references a valid Automation object.

Syntax

IsObject(*expression***)**

The *expression* argument can be any expression.

Remarks

IsObject returns **True** if *expression* is a variable of **Object** subtype or a user-defined object; otherwise, it returns **False**.

IsReady Property

Description

Returns **True** if the specified drive is ready; **False** if it is not.

Syntax

object.**IsReady**

The *object* is always a **Drive** object.

Remarks

For removable-media drives and CD-ROM drives, **IsReady** returns **True** only when the appropriate media is inserted and ready for access.

The following code illustrates the use of the **IsReady** property:

```
Sub ShowDriveInfo(drvpath)
   Dim fs, d, s, t
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(drvpath)
   Select Case d.DriveType
      Case 0: t = "Unknown"
      Case 1: t = "Removable"
      Case 2: t = "Fixed"
      Case 3: t = "Network"
      Case 4: t = "CD-ROM"
      Case 5: t = "RAM Disk"
   End Select
   s = "Drive " & d.DriveLetter & ": - " & t
   If d.IsReady Then
      s = s & vbCrLf & "Drive is Ready."
   Else
      s = s & vbCrLf & "Drive is not Ready."
   End If
   MsgBox s
End Sub
```

IsRootFolder Property

Description

Returns **True** if the specified folder is the root folder; **False** if it is not.

Syntax

object.**IsRootFolder**

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **IsRootFolder** property:

```
Dim fs
Set fs = CreateObject("Scripting.FileSystemObject")
Sub DisplayLevelDepth(pathspec)
   Dim f, n
   Set f = fs.GetFolder(pathspec)
   If f.IsRootFolder Then
      MsgBox "The specified folder is the root folder."
   Else
      Do Until f.IsRootFolder
         Set f = f.ParentFolder
         n = n + 1
      Loop
      MsgBox "The specified folder is nested " & n & " levels deep."
   End If
End Sub
```

Item Property

Description

Sets or returns an *item* for a specified *key* in a **Dictionary** object. For collections, returns an *item* based on the specified *key*. Read/write.

Syntax

object.**Item(***key***)** [= *newitem*]

The **Item** property has the following parts:

Part	Description
object	Required. Always the name of a collection or **Dictionary** object.
key	Required. *Key* associated with the *item* being retrieved or added.
newitem	Optional. Used for **Dictionary** object only; no application for collections. If provided, *newitem* is the new value associated with the specified *key*.

Remarks

If *key* is not found when changing an *item*, a new *key* is created with the specified *newitem*. If *key* is not found when attempting to return an existing item, a new *key* is created and its corresponding item is left empty.

Items Method

Description

Returns an array containing all the items in a **Dictionary** object.

Syntax

object.**Items**

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **Items** method:

```
Dim a, d, i            'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"    'Add some  keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.Items            'Get the items
For i = 0 To d.Count -1 'Iterate the array
   Print a(i)          'Print item
Next
```

Join Function

Description

Returns a string created by joining a number of substrings contained in an array.

Syntax

Join(*list*[, *delimiter*]**)**

The **Join** function syntax has these parts:

Part	Description
list	Required. One-dimensional array containing substrings to be joined.
delimiter	Optional. String character used to separate the substrings in the returned string. If omitted, the space character (" ") is used. If *delimiter* is a zero-length string, all items in the list are concatenated with no delimiters.

Key Property

Description

Sets a *key* in a **Dictionary** object.

Syntax

object.**Key(***key***)** = *newkey*

The **Key** property has the following parts:

Part	Description
object	Required. Always the name of a **Dictionary** object.
key	Required. *Key* value being changed.
newkey	Required. New value that replaces the specified *key*.

Remarks

If *key* is not found when changing a *key*, a run-time error will occur.

Keys Method

Description

Returns an array containing all existing keys in a **Dictionary** object.

Syntax

object.**Keys**

The *object* is always the name of a **Dictionary** object.

Remarks

The following code illustrates use of the **Keys** method:

```
Dim a, d, i            'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"    'Add some  keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
a = d.keys             'Get the keys
For i = 0 To d.Count -1 'Iterate the array
   Print a(i)          'Print key
Next
```

LBound Function

Description

Returns the smallest available subscript for the indicated dimension of an array.

Syntax

LBound(*arrayname***[,** *dimension***])**

The **LBound** function syntax has these parts:

Part	Description
arrayname	Name of the array variable; follows standard variable naming conventions.
dimension	Whole number indicating which dimension's lower bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If *dimension* is omitted, 1 is assumed.

Remarks

The **LBound** function is used with the **UBound** function to determine the size of an array. Use the **UBound** function to find the upper limit of an array dimension.
The default lower bound for any dimension is always 0.

LCase Function

Description

Returns a string that has been converted to lowercase.

Syntax

LCase(*string***)**
The *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

Remarks

Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.

Left Function

Description

Returns a specified number of characters from the left side of a string.

Syntax

Left(*string***,** *length***)**
The **Left** function syntax has these arguments:

Part	Description
string	String expression from which the leftmost characters are returned. If *string* contains **Null**, **Null** is returned.
length	Numeric expression indicating how many characters to return. If 0, a zero-length string("") is returned. If greater than or equal to the number of characters in *string*, the entire string is returned.

Remarks

To determine the number of characters in *string*, use the **Len** function.
Note The **LeftB** function is used with byte data contained in a string. Instead of specifying the number of characters to return, *length* specifies the number of bytes.

Len Function

Description

Returns the number of characters in a string or the number of bytes required to store a variable.

Syntax

Len(*string* | *varname***)**
The **Len** function syntax has these parts:

Part	Description
string	Any valid string expression. If *string* contains **Null**, **Null** is returned.
varname	Any valid variable name. If *varname* contains **Null**, **Null** is returned.

Remarks

Note The LenB function is used with byte data contained in a string. Instead of returning the number of characters in a string, LenB returns the number of bytes used to represent that string.

Line Property

Description

Read-only property that returns the current line number in a **TextStream** file.

Syntax

*object***.Line**
The *object* is always the name of a **TextStream** object.

Remarks

After a file is initially opened and before anything is written, **Line** is equal to 1.

LoadPicture Function

Description

Syntax

Minute(*time***)**
The *time* argument is any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

Mod Operator

Description

Used to divide two numbers and return only the remainder.

Syntax

result **=** *number1* **Mod** *number2*
The **Mod** operator syntax has these parts:

Part	Description
result	Any numeric variable.
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

The modulus, or remainder, operator divides *number1* by *number2* (rounding floating-point numbers to integers) and returns only the remainder as *result*. For example, in the following expression, A (which is *result*) equals 5.

```
A = 19 Mod 6.7
```

If any expression is **Null**, *result* is also **Null**. Any expression that is **Empty** is treated as 0.

Month Function

Description

Returns a whole number between 1 and 12, inclusive, representing the month of the year.

Syntax

Month(*date***)**
The *date* argument is any expression that can represent a date. If *date* contains **Null**, **Null** is returned.

MonthName Function

Description

Returns a string indicating the specified month.

Syntax

MonthName(*month***[,** *abbreviate***])**
The **MonthName** function syntax has these parts:

Part	Description
month	Required. The numeric designation of the month. For example, January is 1, February is 2, and so on.
abbreviate	Optional. Boolean value that indicates if the month name is to be abbreviated. If omitted, the default is **False**, which means that the month name is not abbreviated.

Move Method

Description

Moves a specified file or folder from one location to another.

Returns a picture object. Available only on 32-bit platforms.

Syntax

LoadPicture(*picturename***)**
The *picturename* argument is a string expression that indicates the name of the picture file to be loaded.

Remarks

Graphics formats recognized by **LoadPicture** include bitmap (.bmp) files, icon (.ico) files, run-length encoded (.rle) files, metafile (.wmf) files, enhanced metafiles (.emf), GIF (.gif) files, and JPEG (.jpg) files.

Log Function

Description

Returns the natural logarithm of a number.

Syntax

Log(*number***)**
The *number* argument can be any valid numeric expression greater than 0.

Remarks

The natural logarithm is the logarithm to the base *e*. The constant *e* is approximately 2.718282. You can calculate base-*n* logarithms for any number *x* by dividing the natural logarithm of *x* by the natural logarithm of *n* as follows:

Log*n*(*x*) = Log(*x*) / Log(*n*)

The following example illustrates a custom **Function** that calculates base-10 logarithms:

```
Function Log10(X)
    Log10 = Log(X) / Log(10)
End Function
```

LTrim Function

Description

Returns a copy of a string without leading spaces (**LTrim**), trailing spaces (**RTrim**), or both leading and trailing spaces (**Trim**).

Syntax

LTrim(*string***)**
RTrim(*string***)**
Trim(*string***)**
The *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

Mid Function

Description

Returns a specified number of characters from a string.

Syntax

Mid(*string***,** *start***[,** *length***])**
The **Mid** function syntax has these arguments:

Part	Description
string	String expression from which characters are returned. If *string* contains **Null**, **Null** is returned.
start	Character position in *string* at which the part to be taken begins. If *start* is greater than the number of characters in *string*, **Mid** returns a zero-length string ("").
length	Number of characters to return. If omitted or if there are fewer than *length* characters in the text (including the character at *start*), all characters from the *start* position to the end of the string are returned.

Remarks

To determine the number of characters in *string*, use the **Len** function.
Note The **MidB** function is used with byte data contained in a string. Instead of specifying the number of characters, the arguments specify numbers of bytes.

Minute Function

Description

Returns a whole number between 0 and 59, inclusive, representing the minute of the hour.

Syntax

*object***.Move** *destination*
The **Move** method syntax has these parts:

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
destination	Required. Destination where the file or folder is to be moved. Wildcard characters are not allowed.

Remarks

The results of the **Move** method on a **File** or **Folder** are identical to operations performed using **FileSystemObject.MoveFile** or **FileSystemObject.MoveFolder**. You should note, however, that the alternative methods are capable of moving multiple files or folders.

MoveFile Method

Description

Moves one or more files from one location to another.

Syntax

*object***.MoveFile** *source*, *destination*
The **MoveFile** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
source	Required. The path to the file or files to be moved. The *source* argument string can contain wildcard characters in the last path component only.
destination	Required. The path where the file or files are to be moved. The *destination* argument can't contain wildcard characters.

Remarks

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is assumed to be the name of a destination file to create. In either case, three things can happen when an individual file is moved:

- If *destination* does not exist, the file gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any files. The **MoveFile** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.

Important This method allows moving files between volumes only if supported by the operating system.

MoveFolder Method

Description

Moves one or more folders from one location to another.

Syntax

*object***.MoveFolder** *source*, *destination*
The **MoveFolder** method syntax has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
source	Required. The path to the folder or folders to be moved. The *source* argument string can contain wildcard characters in the last path component only.
destination	Required. The path where the folder or folders are to be moved. The *destination* argument can't contain wildcard characters.

Remarks

If *source* contains wildcards or *destination* ends with a path separator (\), it is assumed that *destination* specifies an existing folder in which to move the matching files. Otherwise, *destination* is

assumed to be the name of a destination folder to create. In either case, three things can happen when an individual folder is moved:

- If *destination* does not exist, the folder gets moved. This is the usual case.
- If *destination* is an existing file, an error occurs.
- If *destination* is a directory, an error occurs.

An error also occurs if a wildcard character that is used in *source* doesn't match any folders. The **MoveFolder** method stops on the first error it encounters. No attempt is made to roll back any changes made before the error occurs.
Important This method allows moving folders between volumes only if supported by the operating system.

MsgBox Function

Description

Displays a message in a dialog box, waits for the user to click a button, and returns a value indicating which button the user clicked.

Syntax

MsgBox(*prompt***[, ***buttons***][, ***title***][, ***helpfile***, ***context***])**
The **MsgBox** function syntax has these arguments:

Part	Description
prompt	String expression displayed as the message in the dialog box. The maximum length of *prompt* is approximately 1024 characters, depending on the width of the characters used. If *prompt* consists of more than one line, you can separate the lines using a carriage return character (**Chr**(13)), a linefeed character (**Chr**(10)), or carriage return–linefeed character combination (**Chr**(13) **&** **Chr**(10)) between each line.
buttons	Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. See Settings section for values. If omitted, the default value for *buttons* is 0.
title	String expression displayed in the title bar of the dialog box. If you omit *title*, the application name is placed in the title bar.
helpfile	String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If *helpfile* is provided, *context* must also be provided. Not available on 16-bit platforms.
context	Numeric expression that identifies the Help context number assigned by the Help author to the appropriate Help topic. If *context* is provided, *helpfile* must also be provided. Not available on 16-bit platforms.

Settings

The *buttons* argument settings are:

Constant	Value	Description
vbOKOnly	0	Display **OK** button only.
vbOKCancel	1	Display **OK** and **Cancel** buttons.
vbAbortRetryIgnore	2	Display **Abort**, **Retry**, and **Ignore** buttons.
vbYesNoCancel	3	Display **Yes**, **No**, and **Cancel** buttons.
vbYesNo	4	Display **Yes** and **No** buttons.
vbRetryCancel	5	Display **Retry** and **Cancel** buttons.
vbCritical	16	Display **Critical Message** icon.
vbQuestion	32	Display **Warning Query** icon.
vbExclamation	48	Display **Warning Message** icon.
vbInformation	64	Display **Information Message** icon.

vbDefaultButton1	0	First button is default.
vbDefaultButton2	256	Second button is default.
vbDefaultButton3	512	Third button is default.
vbDefaultButton4	768	Fourth button is default.
vbApplicationModal	0	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box.

The first group of values (0–5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512, 768) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the argument *buttons*, use only one number from each group.

Return Values

The **MsgBox** function has the following return values:

Constant	Value	Button
vbOK	1	**OK**
vbCancel	2	**Cancel**
vbAbort	3	**Abort**
vbRetry	4	**Retry**
vbIgnore	5	**Ignore**
vbYes	6	**Yes**
vbNo	7	**No**

Remarks

When both *helpfile* and *context* are provided, the user can press **F1** to view the Help topic corresponding to the context.
If the dialog box displays a **Cancel** button, pressing the **ESC** key has the same effect as clicking **Cancel**. If the dialog box contains a **Help** button, context-sensitive Help is provided for the dialog box. However, no value is returned until one of the other buttons is clicked.

Muliplication Operator (*)

Description

Used to multiply two numbers.

Syntax

result **=** *number1******number2*
The * operator syntax has these parts:

Part	Description
result	Any numeric variable.
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

If one or both expressions are **Null** expressions, *result* is **Null**. If an expression is **Empty**, it is treated as if it were 0.

Name Property

Description

Sets or returns the name of a specified file or folder. Read/write.

Syntax

object.**Name** [**=** *newname*]
The **Name** property has these parts:

Part	Description
object	Required. Always the name of a **File** or **Folder** object.
newname	Optional. If provided, *newname* is the new name of the specified *object*.

Remarks

The following code illustrates the use of the **Name** property:

```
Sub ShowFileAccessInfo(filespec)
  Dim fs, f, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.GetFile(filespec)
  s = f.Name & " on Drive " & UCase(f.Drive) & vbCrLf
  s = s & "Created: " & f.DateCreated & vbCrLf
  s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
  s = s & "Last Modified: " & f.DateLastModified
  MsgBox s, 0, "File Access Info"
End Sub
```

Negation Operator (-)

Description

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax 1

result **=** *number1***-***number2*

Syntax 2

-*number*
The **-** operator syntax has these parts:

Part	Description
result	Any numeric variable.
number	Any numeric expression.
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

In Syntax 1, the - operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator to indicate the negative value of an expression.
If one or both expressions are **Null** expressions, *result* is **Null**. If an expression is **Empty**, it is treated as if it were 0.

Not Operator

Description

Used to perform logical negation on an expression.

Syntax

result **= Not** *expression*
The **Not** operator syntax has these parts:

Part	Description
result	Any numeric variable.
expression	Any expression.

Remarks

The following table illustrates how *result* is determined:

If ***expression*** **is**	**Then** ***result*** **is**

True	**False**
False	**True**
Null	**Null**

In addition, the **Not** operator inverts the bit values of any variable and sets the corresponding bit in *result* according to the following table:

Bit in ***expression***	Bit in ***result***
0	1
1	0

Now Function

Description

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax 1

result **=** *number1***-***number2*

Syntax 2

-*number*
The **-** operator syntax has these parts:

Part	Description
result	Any numeric variable.
number	Any numeric expression.
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

In Syntax 1, the - operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator to indicate the negative value of an expression.
If one or both expressions are **Null** expressions, *result* is **Null**. If an expression is **Empty**, it is treated as if it were 0.

Nothing

Description

The **Nothing** keyword in VBScript is used to disassociate an object variable from any actual object. Use the **Set** statement to assign **Nothing** to an object variable. For example:

```
Set MyObject = Nothing
```

Several object variables can refer to the same actual object. When **Nothing** is assigned to an object variable, that variable no longer refers to any actual object. When several object variables refer to the same object, memory and system resources associated with the object to which the variables refer are released only after all of them have been set to **Nothing**, either explicitly using **Set**, or implicitly after the last object variable set to **Nothing** goes out of scope.

Null

Description

The **Null** keyword is used to indicate that a variable contains no valid data. This is not the same thing as **Empty**.

Number Property

Description

Returns or sets a numeric value specifying an error. **Number** is the **Err** object's default property.

Syntax

object.**Number** [= *errornumber*]
The **Number** property syntax has these parts:

Part	Description
object	Always the **Err** object.
errornumber	An integer representing a VBScript error number or an SCODE error value.

Remarks
When returning a user-defined error from an Automation object, set **Err.Number** by adding the number you selected as an error code to the constant **vbObjectError**. For example, you use the following code to return the number 1051 as an error code:

```
Err.Raise Number:= vbObjectError + 1051, Source:= "SomeClass"
```

Oct Function
Description
Returns a string representing the octal value of a number.
Syntax
Oct(*number*)
The *number* argument is any valid expression.
Remarks
If *number* is not already a whole number, it is rounded to the nearest whole number before being evaluated.

If *number* is	Oct returns
Null	**Null**.
Empty	Zero (0).
Any other number	Up to 11 octal characters,

You can represent octal numbers directly by preceding numbers in the proper range with &O. For example, &O10 is the octal notation for decimal 8.

On Error Statement
Description
Enables error-handling.
Syntax
On Error Resume Next
Remarks
If you don't use an **On Error Resume Next** statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.
On Error Resume Next causes execution to continue with the statement immediately following the statement that caused the run-time error, or with the statement immediately following the most recent call out of the procedure containing the **On Error Resume Next** statement. This allows execution to continue despite a run-time error. You can then build the error-handling routine inline within the procedure. An **On Error Resume Next** statement becomes inactive when another procedure is called, so you should execute an **On Error Resume Next** statement in each called routine if you want inline error handling within that routine.

OpenAsTextStream Method
Description
Opens a specified file and returns a **TextStream** object that can be used to read from, write to, or append to the file.

Syntax
object.**OpenTextFile(***filename*[**,** *iomode*[**,** *create*[**,** *format*]]]**)**
The **OpenTextFile** method has these parts:

Part	Description
object	Required. Always the name of a **FileSystemObject**.
filename	Required. String expression that identifies the file to open.
iomode	Optional. Indicates input/output mode. Can be one of two constants, either **ForReading** or **ForAppending**.
create	Optional. Boolean value that indicates whether a new file can be created if the specified *filename* doesn't exist. The value is **True** if a new file is created; **False** if it isn't created. The default is **False**.
format	Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Settings
The *iomode* argument can have either of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default.
TristateTrue	-1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks
The following code illustrates the use of the **OpenTextFile** method to open a file for appending text:

```
Sub OpenTextFileTest
  Const ForReading = 1, ForWriting = 2, ForAppending = 3
  Dim fs, f
  Set fs = CreateObject("Scripting.FileSystemObject")
  Set f = fs.OpenTextFile("c:\testfile.txt", ForAppending,TristateFalse)
  f.Write "Hello world!"
  f.Close
End Sub
```

Operator Precedence
Description
When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence. Parentheses can be used to override the order of

Syntax
object.**OpenAsTextStream([***iomode*, [*format*]**])**
The **OpenAsTextStream** method syntax has these parts:

Part	Description
object	Required. Always the name of a **File** object.
iomode	Optional. Indicates input/output mode. Can be one of three constants: **ForReading**, **ForWriting**, or **ForAppending**.
format	Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

Settings
The *iomode* argument can have any of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

The *format* argument can have any of the following settings:

Constant	Value	Description
TristateUseDefault	-2	Opens the file using the system default.
TristateTrue	-1	Opens the file as Unicode.
TristateFalse	0	Opens the file as ASCII.

Remarks
The **OpenAsTextStream** method provides the same functionality as the **OpenTextFile** method of the **FileSystemObject**. In addition, the **OpenAsTextStream** method can be used to write to a file.
The following code illustrates the use of the **OpenAsTextStream** method:

```
Sub TextStreamTest
  Const ForReading = 1, ForWriting = 2, ForAppending = 3
  Const TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0
  Dim fs, f, ts, s
  Set fs = CreateObject("Scripting.FileSystemObject")
  fs.CreateTextFile "test1.txt"          'Create a file
  Set f = fs.GetFile("test1.txt")
  Set ts = f.OpenAsTextStream(ForWriting, TristateUseDefault)
  ts.Write "Hello World"
  ts.Close
  Set ts = f.OpenAsTextStream(ForReading, TristateUseDefault)
  s = ts.ReadLine
  MsgBox s
  ts.Close
End Sub
```

OpenTextFile Method
Description
Opens a specified file and returns a **TextStream** object that can be used to read from or append to the file.

precedence and force some parts of an expression to be evaluated before other parts. Operations within parentheses are always performed before those outside. Within parentheses, however, normal operator precedence is maintained.
When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence:

Arithmetic	Comparison	Logical
Exponentiation (**^**)	Equality (**=**)	**Not**
Negation (**-**)	Inequality (**<>**)	**And**
Multiplication and division (*****, **/**)	Less than (**<**)	**Or**
Integer division (****)	Greater than (**>**)	**Xor**
Modulus arithmetic (**Mod**)	Less than or equal to (**<=**)	**Eqv**
Addition and subtraction (**+**, **-**)	Greater than or equal to (**>=**)	**Imp**
String concatenation (**&**)	**Is**	**&**

When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. Likewise, when addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.
The string concatenation operator (**&**) is not an arithmetic operator, but in precedence it does fall after all arithmetic operators and before all comparison operators. The **Is** operator is an object reference comparison operator. It does not compare objects or their values; it checks only to determine if two object references refer to the same object.

Option Explicit Statement
Description
Used at script level to force explicit declaration of all variables in that script.
Syntax
Option Explicit
Remarks
If used, the **Option Explicit** statement must appear in a script before any procedures.
When you use the **Option Explicit** statement, you must explicitly declare all variables using the **Dim**, **Private**, **Public**, or **ReDim** statements. If you attempt to use an undeclared variable name, an error occurs.
Tip Use **Option Explicit** to avoid incorrectly typing the name of an existing variable or to avoid confusion in code where the scope of the variable is not clear.

Or Operator
Description
Used to perform a logical disjunction on two expressions.
Syntax
result **=** *expression1* **Or** *expression2*
The **Or** operator syntax has these parts:

Part	Description
result	Any numeric variable.
expression1	Any expression.
expression2	Any expression.

Remarks
If either or both expressions evaluate to **True**, *result* is **True**. The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	Then *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**

The **Or** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	Then *result* is
0	0	0
0	1	1
1	0	1
1	1	1

ParentFolder Property

Description

Returns the folder object for the parent of the specified file or folder. Read-only.

Syntax

object.**ParentFolder**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ParentFolder** property with a file:

```
Sub ShowFileAccessInfo(filespec)
   Dim fs, f, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFile(filespec)
   s = UCase(f.Name) & " in " & UCase(f.ParentFolder) & vbCrLf
   s = s & "Created: " & f.DateCreated & vbCrLf
   s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
   s = s & "Last Modified: " & f.DateLastModified
   MsgBox s, 0, "File Access Info"
End Sub
```

Path Property

Description

Returns the path for a specified file, folder, or drive.

Syntax

object.**Path**

The *object* is always a **File**, **Folder**, or **Drive** object.

Remarks

For drive letters, the root drive is not included. For example, the path for the C drive is C:, not C:\.

The following code illustrates the use of the **Path** property with a **File** object:

```
Sub ShowFileAccessInfo(filespec)
   Dim fs, d, f, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFile(filespec)
   s = UCase(f.Path) & vbCrLf
```



```
   s = s & "Created: " & f.DateCreated & vbCrLf
   s = s & "Last Accessed: " & f.DateLastAccessed & vbCrLf
   s = s & "Last Modified: " & f.DateLastModified
   MsgBox s, 0, "File Access Info"
End Sub
```

Private Statement

Description

Used at script level to declare private variables and allocate storage space.

Syntax

Private *varname*[([*subscripts*])][**,** *varname*[([*subscripts*])]] **. . .**

The **Private** statement syntax has these parts:

Part	Description
varname	Name of the variable; follows standard variable naming conventions.
subscripts	Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: *upper* [**,** *upper*] **. . .** The lower bound of an array is always zero.

Remarks

Private variables are available only to the script in which they are declared.

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.

You can also use the **Private** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

When variables are initialized, a numeric variable is initialized to 0 and a string is initialized to a zero-length string ("").

Tip When you use the **Private** statement in a procedure, you generally put the **Private** statement at the beginning of the procedure.

Public Statement

Description

Used at script level to declare public variables and allocate storage space.

Syntax

Public *varname*[([*subscripts*])][**,** *varname*[([*subscripts*])]] **. . .**

The **Public** statement syntax has these parts:

Part	Description
varname	Name of the variable; follows standard variable naming conventions.
subscripts	Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: *upper* [**,***upper*] **. . .** The lower bound of an array is always zero.

Remarks

Variables declared using the **Public** statement are available to all procedures in all scripts in all projects.

A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.

You can also use the **Public** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

When variables are initialized, a numeric variable is initialized to 0 and a string is initialized to a zero-length string ("").

Raise Method

Description

Generates a run-time error.

Syntax

object.**Raise(***number***,** *source***,** *description***,** *helpfile***,** *helpcontext***)**

The **Raise** method has these parts:

Part	Description
object	Always the **Err** object.
number	A **Long** integer subtype that identifies the nature of the error. VBScript errors (both VBScript-defined and user-defined errors) are in the range 0–65535.
source	A string expression naming the object or application that originally generated the error. When setting this property for an Automation object, use the form *project.class*. If nothing is specified, the programmatic ID of the current VBScript project is used.
description	A string expression describing the error. If unspecified, the value in *number* is examined. If it can be mapped to a VBScript run-time error code, a string provided by VBScript is used as *description*. If there is no VBScript error corresponding to *number*, a generic error message is used.
helpfile	The fully qualified path to the Help file in which help on this error can be found. If unspecified, VBScript uses the fully qualified drive, path, and file name of the VBScript Help file.
helpcontext	The context ID identifying a topic within *helpfile* that provides help for the error. If omitted, the VBScript Help file context ID for the error corresponding to the *number* property is used, if it exists.

Remarks

All the arguments are optional except *number*. If you use **Raise**, however, without specifying some arguments, and the property settings of the **Err** object contain values that have not been cleared, those values become the values for your error.

When setting the *number* property to your own error code in an Automation object, you add your error code number to the constant **vbObjectError**. For example, to generate the error number 1050, assign **vbObjectError** + 1050 to the *number* property.

Randomize Statement

Description

Initializes the random-number generator.

Syntax

Randomize [*number*]

The *number* argument can be any valid numeric expression.

Remarks

Randomize uses *number* to initialize the **Rnd** function's random-number generator, giving it a new seed value. If you omit *number*, the value returned by the system timer is used as the new seed value.

If **Randomize** is not used, the **Rnd** function (with no arguments) uses the same number as a seed the first time it is called, and thereafter uses the last generated number as a seed value.

Note To repeat sequences of random numbers, call **Rnd** with a negative argument immediately before using **Randomize** with a numeric argument. Using **Randomize** with the same value for *number* does not repeat the previous sequence.

Read Method

Description

Reads a specified number of characters from a **TextStream** file and returns the resulting string.

Syntax

object.**Read(***characters***)**

The **Read** method syntax has these parts:

Part	Description
object	Required. Always the name of a **TextStream** object.
characters	Required. Number of characters you want to read from the file.

ReadAll Method

Description

Reads an entire **TextStream** file and returns the resulting string.

Syntax

object.**ReadAll**

The *object* is always the name of a **TextStream** object.

Remarks

For large files, using the **ReadAll** method wastes memory resources. Other techniques should be used to input a file, such as reading a file line by line.

ReadLine Method

Description

Reads an entire line (up to, but not including, the newline character) from a **TextStream** file and returns the resulting string.

Syntax

object.**ReadLine**

The *object* argument is always the name of a **TextStream** object.

ReDim Statement

Description

Used at procedure level to declare dynamic-array variables and allocate or reallocate storage space.

Syntax

ReDim [**Preserve**] *varname*(*subscripts*) [**,** *varname*(*subscripts*)] **. . .**

The **ReDim** statement syntax has these parts:

Part	Description
Preserve	Preserves the data in an existing array when you change the size of the last dimension.
varname	Name of the variable; follows standard variable naming conventions.
subscripts	Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: *upper* [**,***upper*] **. . .** The lower bound of an array is always zero.

Remarks

The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts). You can use the **ReDim** statement repeatedly to change the number of elements and dimensions in an array.

If you use the **Preserve** keyword, you can resize only the last array dimension, and you can't change the number of dimensions at all. For example, if your array has only one dimension, you can resize that dimension because it is the last and only dimension. However, if your array has two or more dimensions, you can change the size of only the last dimension and still preserve the contents of the array. The following example shows how you can increase the size of the last dimension of a dynamic array without erasing any existing data contained in the array.

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```

Caution If you make an array smaller than it was originally, data in the eliminated elements is lost.

When variables are initialized, a numeric variable is initialized to 0 and a string variable is initialized to a zero-length string (""). A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.

Rem Statement

Description

Used to include explanatory remarks in a program.

Syntax

Rem *comment*
or
' *comment*
The *comment* argument is the text of any comment you want to include. After the **Rem** keyword, a space is required before *comment.*

Remarks

As shown in the syntax section, you can use an apostrophe (') instead of the **Rem** keyword. If the **Rem** keyword follows other statements on a line, it must be separated from the statements by a colon. However, when you use an apostrophe, the colon is not required after other statements.

Remove Method

Description

Removes a key, item pair from a **Dictionary** object.

Syntax

object.**Remove(***key***)**
The **Remove** method syntax has these parts:

Part	Description
object	Required. Always the name of a **Dictionary** object.
key	Required. *Key* associated with the key, item pair you want to remove from the **Dictionary** object.

Remarks

An error occurs if the specified key, item pair does not exist.
The following code illustrates use of the **Remove** method:

```
Dim a, d, i             'Create some variables
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"     'Add some keys and items
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
...
a = d.Remove("b")       'Remove second pair
```

RemoveAll Method

Description

Used at procedure level to declare dynamic-array variables and allocate or reallocate storage space.

Syntax

ReDim [**Preserve**] *varname*(*subscripts*) [**,** *varname*(*subscripts*)] **. . .**
The **ReDim** statement syntax has these parts:

Part	Description
Preserve	Preserves the data in an existing array when you change the size of the last dimension.
varname	Name of the variable; follows standard variable naming conventions.
subscripts	Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: *upper* [**,***upper*] **. . .** The lower bound of an array is always zero.

Remarks

The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts). You can use the **ReDim** statement repeatedly to change the number of elements and dimensions in an array.

If you use the **Preserve** keyword, you can resize only the last array dimension, and you can't change the number of dimensions at all. For example, if your array has only one dimension, you can resize that dimension because it is the last and only dimension. However, if your array has two or more dimensions, you can change the size of only the last dimension and still preserve the contents of the array. The following example shows how you can increase the size of the last dimension of a dynamic array without erasing any existing data contained in the array.

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```

Caution If you make an array smaller than it was originally, data in the eliminated elements is lost.
When variables are initialized, a numeric variable is initialized to 0 and a string variable is initialized to a zero-length string (""). A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**.

Replace Function

Description

Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax

Replace(*expression***,** *find***,** *replacewith***[,** *start***[,** *count***[,** *compare***]]])**
The **Replace** function syntax has these parts:

Part	Description
expression	Required. String expression containing substring to replace.
find	Required. Substring being searched for.
replacewith	Required. Replacement substring.
start	Optional. Position within *expression* where substring search is to begin. If omitted, 1 is assumed.
count	Optional. Number of substring substitutions to perform. If omitted, the default value is -1, which means make all possible substitutions.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values.

Settings

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values

Replace returns the following values:

If	Replace returns
expression is zero-length	Zero-length string ("").
expression is **Null**	An error.
find is zero-length	Copy of *expression.*
replacewith is zero-length	Copy of *expression* with all occurences of *find* removed.
start > **Len(***expression***)**	Zero-length string.
count is 0	Copy of *expression.*

Remarks

The return value of the **Replace** function is a string, with substitutions made, that begins at the position specified by *start* and and concludes at the end of the *expression* string. It is not a copy of the original string from start to finish.

RGB Function

Description

Returns a whole number representing an RGB color value.

Syntax

RGB(*red, green, blue***)**
The **RGB** function has these parts:

Part	Description
red	Required. Number in the range 0-255 representing the red component of the color.
green	Required. Number in the range 0-255 representing the green component of the color.
blue	Required. Number in the range 0-255 representing the blue component of the color.

Remarks

Application methods and properties that accept a color specification expect that specification to be a number representing an RGB color value. An RGB color value specifies the relative intensity of red, green, and blue to cause a specific color to be displayed.
The low-order byte contains the value for red, the middle byte contains the value for green, and the high-order byte contains the value for blue.
For applications that require the byte order to be reversed, the following function will provide the same information with the bytes reversed:

```
Function RevRGB(red, green, blue)
   RevRGB= CLng(blue + (green * 256) + (red * 65536))
End Function
```

The value for any argument to RGB that exceeds 255 is assumed to be 255.

Right Function

Description

Returns a specified number of characters from the right side of a string.

Syntax

Right(*string***,** *length***)**
The **Right** function syntax has these arguments:

Part	Description
string	String expression from which the rightmost characters are returned. If *string* contains **Null**, **Null** is returned.
length	Numeric expression indicating how many characters to return. If 0, a zero-length string is returned. If greater than or equal to the number of characters in *string*, the entire string is returned.

Remarks

To determine the number of characters in *string*, use the **Len** function.
Note The **RightB** function is used with byte data contained in a string. Instead of specifying the number of characters to return, *length* specifies the number of bytes.

Rnd Function

Description

Returns a random number.

Syntax

Rnd[(*number***)]**
The *number* argument can be any valid numeric expression.

Remarks

The **Rnd** function returns a value less than 1 but greater than or equal to 0.
The value of *number* determines how **Rnd** generates a random number:

If *number* is	Rnd generates
Less than zero	The same number every time, using *number* as the seed.
Greater than zero	The next random number in the sequence.
Equal to zero	The most recently generated number.
Not supplied	The next random number in the sequence.

For any given initial seed, the same number sequence is generated because each successive call to the **Rnd** function uses the previous number as a seed for the next number in the sequence.
Before calling **Rnd**, use the **Randomize** statement without an argument to initialize the random-number generator with a seed based on the system timer.
To produce random integers in a given range, use this formula:

```
Int((upperbound - lowerbound + 1) * Rnd + lowerbound)
```

Here, *upperbound* is the highest number in the range, and *lowerbound* is the lowest number in the range.
Note To repeat sequences of random numbers, call **Rnd** with a negative argument immediately before using **Randomize** with a numeric argument. Using **Randomize** with the same value for *number* does not repeat the previous sequence

RootFolder Property

Description

Returns a **Folder** object representing the root folder of a specified drive. Read-only.

Syntax

object.**RootFolder**
The *object* is always a **Drive** object.

Remarks

All the files and folders contained on the drive can be accessed using the returned **Folder** object.

Round Function

Description

Returns a number rounded to a specified number of decimal places.

Syntax

Round(*expression***[,** *numdecimalplaces***])**
The **Round** function syntax has these parts:

Part	Description
expression	Required. Numeric expression being rounded.
numdecimalplaces	Optional. Number indicating how many places to the right of the decimal are included in the rounding. If omitted, integers are returned by the **Round** function.

RTrim Function

Description

Returns a copy of a string without leading spaces (**LTrim**), trailing spaces (**RTrim**), or both leading and trailing spaces (**Trim**).

Syntax

LTrim(*string***)**
RTrim(*string***)**
Trim(*string***)**
The *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

ScriptEngine Function

Description

Returns a string representing the scripting language in use.

Syntax

ScriptEngine

Return Values

The **ScriptEngine** function can return any of the following strings:

String	Description
VBScript	Indicates that Microsoft® Visual Basic® Scripting Edition is the current script engine.
JScript	Indicates that Microsoft JScript™ is the current script engine.
VBA	Indicates that Microsoft Visual Basic for Applications is the current script engine.

ScriptEngineBuildVersion Function

Description

Returns the build version number of the script engine in use.

Syntax

ScriptEngineBuildVersion

Remarks

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

ScriptEngineMajorVersion Function

Description

Returns the major version number of the script engine in use.

Syntax

ScriptEngineMajorVersion

Remarks

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

ScriptEngineMinorVersion Function

Description

Returns the minor version number of the script engine in use.

Syntax

ScriptEngineMinorVersion

Remarks

The return value corresponds directly to the version information contained in the DLL for the scripting language in use.

Second Function

Description

Returns a whole number between 0 and 59, inclusive, representing the second of the minute.

Syntax

Second(*time***)**
The *time* argument is any expression that can represent a time. If *time* contains **Null**, **Null** is returned.

Select Case Statement

Description

Executes one of several groups of statements, depending on the value of an expression.

Syntax

Select Case *testexpression*
[**Case** *expressionlist-n*
[*statements-n*]] **. . .**
[**Case Else** *expressionlist-n*
[*elsestatements-n*]]
End Select
The **Select Case** statement syntax has these parts:

Part	Description
testexpression	Any numeric or string expression.
expressionlist-n	Required if **Case** appears. Delimited list of one or more expressions.
statements-n	One or more statements executed if *testexpression* matches any part of *expressionlist-n*.
elsestatements	One or more statements executed if *testexpression* doesn't match any of the **Case** clauses.

Remarks

If *testexpression* matches any **Case** *expressionlist* expression, the statements following that **Case** clause are executed up to the next **Case** clause, or for the last clause, up to **End Select**. Control then passes to the statement following **End Select**. If *testexpression* matches an *expressionlist* expression in more than one **Case** clause, only the statements following the first match are executed.
The **Case Else** clause is used to indicate the *elsestatements* to be executed if no match is found between the *testexpression* and an *expressionlist* in any of the other **Case** selections. Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen *testexpression* values. If no **Case** *expressionlist* matches *testexpression* and there is no **Case Else** statement, execution continues at the statement following **End Select**.
Select Case statements can be nested. Each nested **Select Case** statement must have a matching **End Select** statement.

SerialNumber Property

Description

Returns the decimal serial number used to uniquely identify a disk volume.

Syntax

object.**SerialNumber**
The *object* is always a **Drive** object.

Remarks

You can use the **SerialNumber** property to ensure that the correct disk is inserted in a drive with removable media.
The following code illustrates the use of the **SerialNumber** property:

```
Sub ShowDriveInfo(drvpath)
   Dim fs, d, s, t
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutePathName(drvpath)))
   Select Case d.DriveType
      Case 0: t = "Unknown"
      Case 1: t = "Removable"
      Case 2: t = "Fixed"
      Case 3: t = "Network"
```



```
      Case 4: t = "CD-ROM"
      Case 5: t = "RAM Disk"
   End Select
   s = "Drive " & d.DriveLetter & ": - " & t
   s = s & vbCrLf & "SN: " & d.SerialNumber
   MsgBox s
End Sub
```

Set Statement

Description

Assigns an object reference to a variable or property.

Syntax

Set *objectvar* **=** {*objectexpression* | **Nothing**}
The **Set** statement syntax has these parts:

Part	Description
objectvar	Name of the variable or property; follows standard variable naming conventions.
objectexpression	Expression consisting of the name of an object, another declared variable of the same object type, or a function or method that returns an object of the same object type.
Nothing	Discontinues association of *objectvar* with any specific object. Assigning *objectvar* to **Nothing** releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.

Remarks

To be valid, *objectvar* must be an object type consistent with the object being assigned to it.
The **Dim**, **Private**, **Public**, or **ReDim** statements only declare a variable that refers to an object. No actual object is referred to until you use the **Set** statement to assign a specific object.
Generally, when you use **Set** to assign an object reference to a variable, no copy of the object is created for that variable. Instead, a reference to the object is created. More than one object variable can refer to the same object. Because these variables are references to (rather than copies of) the object, any change in the object is reflected in all variables that refer to it.

Sgn Function

Description

Returns an integer indicating the sign of a number.

Syntax

Sgn(*number***)**
The *number* argument can be any valid numeric expression.

Return Values

The **Sgn** function has the following return values:

If *number* is	Sgn returns
Greater than zero	1
Equal to zero	0
Less than zero	-1

Remarks

The sign of the *number* argument determines the return value of the **Sgn** function.

ShareName Property

Description

Returns the network share name for a specified drive.

Syntax

object.**ShareName**
The *object* is always a **Drive** object.

Remarks

If *object* is not a network drive, the **ShareName** property returns a zero-length string ("").
The following code illustrates the use of the **ShareName** property:

```
Sub ShowDriveInfo(drvpath)
   Dim fs, d, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutePathName(drvpath)))
   s = "Drive " & d.DriveLetter & ": - " & d.ShareName
   MsgBox s
End Sub
```

ShortName Property

Description

Returns the network share name for a specified drive.

Syntax

object.**ShareName**
The *object* is always a **Drive** object.

Remarks

If *object* is not a network drive, the **ShareName** property returns a zero-length string ("").
The following code illustrates the use of the **ShareName** property:

```
Sub ShowDriveInfo(drvpath)
   Dim fs, d, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutePathName(drvpath)))
   s = "Drive " & d.DriveLetter & ": - " & d.ShareName
   MsgBox s
End Sub
```

ShortPath Property

Description

Returns the short path used by programs that require the earlier 8.3 file naming convention.

Syntax

object.**ShortPath**
The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **ShortName** property with a **File** object:

```
Sub ShowShortPath(filespec)
   Dim fs, f, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFile(filespec)
   s = "The short path for " & "" & UCase(f.Name)
   s = s & "" & vbCrLf
   s = s & "is: " & "" & f.ShortPath & ""
   MsgBox s, 0, "Short Path Info"
End Sub
```

Sin Function

Description

Returns the sine of an angle.

Syntax

Sin(*number***)** The *number* argument can be any valid numeric expression that expresses an angle in radians.

Remarks

The **Sin** function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse. The result lies in the range -1 to 1. To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Size Property

Description

For files, returns the size, in bytes, of the specified file. For folders, returns the size, in bytes, of all files and subfolders contained in the folder.

Syntax

object.**Size**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Size** property with a **Folder** object:

```
Sub ShowFolderSize(filespec)
   Dim fs, f, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFolder(filespec)
   s = UCase(f.Name) & " uses " & f.size & " bytes."
   MsgBox s, 0, "Folder Size Info"
End Sub
```

Skip Method

Description

Skips a specified number of characters when reading a **TextStream** file.

Syntax

object.**Skip**(*characters*)

The **Skip** method syntax has these parts:

Part	Description
object	Required. Always the name of a **TextStream** object.
characters	Required. Number of characters to skip when reading a file.

Remarks

Skipped characters are discarded.

SkipLine Method

Description

Skips the next line when reading a **TextStream** file.

Syntax

object.**SkipLine**

The *object* is always the name of a **TextStream** object.

Remarks

Skipping a line means reading and discarding all characters in a line up to and including the next newline character.

An error occurs if the file is not open for reading.

Source Property

Description

Returns or sets the name of the object or application that originally generated the error.

Syntax

object.**Source** [= *stringexpression*]

The **Source** property syntax has these parts:

Part	Description
object	Always the **Err** object.
stringexpression	A string expression representing the application that generated the error.

Remarks

The **Source** property specifies a string expression that is usually the class name or programmatic ID of the object that caused the error. Use **Source** to provide your users with information when your code is unable to handle an error generated in an accessed object. For example, if you access Microsoft Excel and it generates a *Division by zero* error, Microsoft Excel sets **Err.Number** to its error code for that error and sets **Source** to Excel.Application. Note that if the error is generated in another object called by Microsoft Excel, Excel intercepts the error and sets **Err.Number** to its own code for *Division by zero*. However, it leaves the other **Err** object (including **Source**) as set by the object that generated the error.

Source always contains the name of the object that originally generated the error — your code can try to handle the error according to the error documentation of the object you accessed. If your error handler fails, you can use the **Err** object information to describe the error to your user, using **Source** and the other **Err** to inform the user which object originally caused the error, its description of the error, and so forth.

When generating an error from code, **Source** is your application's programmatic ID.

Space Function

Description

Returns a string consisting of the specified number of spaces.

Syntax

Space(*number*)

The *number* argument is the number of spaces you want in the string.

Split Function

Description

Returns a zero-based, one-dimensional array containing a specified number of substrings.

Syntax

Split(*expression*[**,** *delimiter*[**,** *count*[**,** *compare*]]])

The **Split** function syntax has these parts:

Part	Description
expression	Required. String expression containing substrings and delimiters. If *expression* is a zero-length string, **Split** returns an empty array, that is, an array with no elements and no data.
delimiter	Optional. String character used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If *delimiter* is a zero-length string, a single-element array containing the entire *expression* string is returned.
count	Optional. Number of substrings to be returned; -1 indicates that all substrings are returned.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values.

Settings

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Sqr Function

Description

Returns the square root of a number.

Syntax

Sqr(*number*)

The *number* argument can be any valid numeric expression greater than or equal to 0.

StrComp Function

Description

Returns a value indicating the result of a string comparison.

Syntax

StrComp(*string1***,** *string2*[**,** *compare*])

The **StrComp** function syntax has these arguments:

Part	Description
string1	Required. Any valid string expression.
string2	Required. Any valid string expression.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating strings. If omitted, a binary comparison is performed. See Settings section for values.

Settings

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values

The **StrComp** function has the following return values:

If	StrComp returns
string1 is less than *string2*	-1
string1 is equal to *string2*	0
string1 is greater than *string2*	1
string1 or *string2* is **Null**	**Null**

String Function

Description

Returns a repeating character string of the length specified.

Syntax

String(*number***,** *character*)

The **String** function syntax has these arguments:

Part	Description
number	Length of the returned string. If *number* contains **Null**, **Null** is returned.
character	Character code specifying the character or string expression whose first character is used to build the return string. If *character* contains **Null**, **Null** is returned.

Remarks

If you specify a number for *character* greater than 255, **String** converts the number to a valid character code using the formula:

character Mod 256

StrReverse Function

Description

Returns a string in which the character order of a specified string is reversed.

Syntax

StrReverse(*string1*)

The *string1* argument is the string whose characters are to be reversed. If *string1* is a zero-length string (""), a zero-length string is returned. If *string1* is **Null**, an error occurs.

Sub Statement

Description

Declares the name, arguments, and code that form the body of a **Sub** procedure.

Syntax

[**Public** | **Private**] **Sub** *name* [(*arglist*)]
   [*statements*]
   [**Exit Sub**]
   [*statements*]
End Sub

The **Sub** statement syntax has these parts:

Part	Description
Public	Indicates that the **Sub** procedure is accessible to all other procedures in all scripts.
Private	Indicates that the **Sub** procedure is accessible only to other procedures in the script where it is declared.
name	Name of the **Sub**; follows standard variable naming conventions.
arglist	List of variables representing arguments that are passed to the **Sub** procedure when it is called. Multiple variables are separated by commas.
statements	Any group of statements to be executed within the body of the **Sub** procedure.

The *arglist* argument has the following syntax and parts:

[**ByVal** | **ByVal**] *varname*[( )]

Part	Description
ByVal	Indicates that the argument is passed by value.
ByRef	Indicates that the argument is passed by reference.
varname	Name of the variable representing the argument; follows standard variable naming conventions.

Remarks

If not explicitly specified using either **Public** or **Private**, **Sub** procedures are public by default, that is, they are visible to all other procedures in your script. The value of local variables in a **Sub** procedure is not preserved between calls to the procedure.

All executable code must be contained in procedures. You can't define a **Sub** procedure inside another **Sub** or **Function** procedure.

The **Exit Sub** statement causes an immediate exit from a **Sub** procedure. Program execution continues with the statement following the statement that called the **Sub** procedure. Any number of **Exit Sub** statements can appear anywhere in a **Sub** procedure.

Like a **Function** procedure, a **Sub** procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a **Function** procedure, which returns a value, a **Sub** procedure can't be used in an expression.

You call a **Sub** procedure using the procedure name followed by the argument list. See the **Call** statement for specific information on how to call **Sub** procedures.

Caution **Sub** procedures can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow.

Variables used in **Sub** procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using **Dim** or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

Caution A procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you have defined at the script level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure, constant or variable, it is assumed that your procedure is referring to that script-level name. Explicitly declare variables to avoid this kind of conflict. You can use an **Option Explicit** statement to force explicit declaration of variables.

SubFolders Property

Description

Returns a **Folders** collection consisting of all folders contained in a specified folder, including those with Hidden and System file attributes set.

Syntax

object.**SubFolders**

The *object* is always a **Folder** object.

Remarks

The following code illustrates the use of the **SubFolders** property:

```
Sub ShowFolderList(folderspec)
   Dim fs, f, f1, s, sf
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFolder(folderspec)
   Set sf = f.SubFolders
   For Each f1 in sf
      s = s & f1.name
      s = s &  vbCrLf
   Next
   MsgBox s
End Sub
```

Subtraction Operator (-)

Description

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax 1

result **=** *number1-number2*

Syntax 2

-*number*

The **-** operator syntax has these parts:

Part	Description
result	Any numeric variable.
number	Any numeric expression.
number1	Any numeric expression.
number2	Any numeric expression.

Remarks

In Syntax 1, the - operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the - operator is used as the unary negation operator to indicate the negative value of an expression.

If one or both expressions are **Null** expressions, *result* is **Null**. If an expression is **Empty**, it is treated as if it were 0.

Tan Function

Description

Returns the tangent of an angle.

Syntax

Tan(*number***)**

The *number* argument can be any valid numeric expression that expresses an angle in radians.

Remarks

Tan takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Time Function

Description

Returns a **Variant** of subtype **Date** indicating the current system time.

Syntax

Time

TextStream Object

Description

Facilitates sequential access to file.

Syntax

TextStream.{*property* | *method*}

The *property* and *method* arguments can be any of the properties and methods associated with the **TextStream** object. Note that in actual usage **TextStream** is replaced by a variable placeholder representing the **TextStream** object returned from the **FileSystemObject**.

Remarks

In the following code, a is the **TextStream** object returned by the **CreateTextFile** method on the **FileSystemObject**:

```
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("c:\testfile.txt", True)
a.WriteLine("This is a test.")
a.Close
```

WriteLine and **Close** are two methods of the **TextStream** Object.

TimeSerial Function

Description

Returns a **Variant** of subtype **Date** containing the time for a specific hour, minute, and second.

Syntax

TimeSerial(*hour***,** *minute***,** *second***)**

The **TimeSerial** function syntax has these arguments:

Part	Description
hour	Number between 0 (12:00 A.M.) and 23 (11:00 P.M.), inclusive, or a numeric expression.
minute	Any numeric expression.
second	Any numeric expression.

Remarks

To specify a time, such as 11:59:59, the range of numbers for each **TimeSerial** argument should be in the accepted range for the unit; that is, 0–23 for hours and 0–59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time. The following example uses expressions instead of absolute time numbers. The **TimeSerial** function returns a time for 15 minutes before (-15) six hours before noon (12 - 6), or 5:45:00 A.M.

```
TimeSerial(12 - 6, -15, 0)
```

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 75 minutes, it is evaluated as one hour and 15 minutes. However, if any single argument is outside the range -32,768 to 32,767, or if the time specified by the three arguments, either directly or by expression, causes the date to fall outside the acceptable range of dates, an error occurs.

TimeValue Function

Description

Returns a **Variant** of subtype **Date** containing the time.

Syntax

TimeValue(*time***)**

The *time* argument is usually a string expression representing a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive. However, *time* can also be any expression that represents a time in that range. If *time* contains **Null**, **Null** is returned.

Remarks

You can enter valid times using a 12-hour or 24-hour clock. For example, "2:24PM" and "14:24" are both valid *time* arguments.

If the *time* argument contains date information, **TimeValue** doesn't return the date information. However, if *time* includes invalid date information, an error occurs.

TotalSize Property

Description

Returns the total space, in bytes, of a drive or network share.

Syntax

object.**TotalSize**

The *object* is always a **Drive** object.

Remarks

The following code illustrates the use of the **TotalSize** property:

```
Sub ShowSpaceInfo(drvpath)
   Dim fs, d, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutePathName(drvpath)))
   s = "Drive " & d.DriveLetter & ":"
   s = s & vbCrLf
   s = s & "Total Size: " & FormatNumber(d.TotalSize/1024, 0) & " Kbytes"
   s = s & vbCrLf
   s = s & "Available: " & FormatNumber(d.AvailableSpace/1024, 0) & " Kbytes"
   MsgBox s
End Sub
```

Trim Function

Description

Returns a copy of a string without leading spaces (**LTrim**), trailing spaces (**RTrim**), or both leading and trailing spaces (**Trim**).

Syntax

LTrim(*string***)**

RTrim(*string***)**

Trim(*string***)**

The *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

True

Description

The **True** keyword has a value equal to -1.

Type Property

Description

Returns information about the type of a file or folder. For example, for files ending in .TXT, "Text Document" is returned.

Syntax

object.**Type**

The *object* is always a **File** or **Folder** object.

Remarks

The following code illustrates the use of the **Type** property to return a folder type. In this example, try providing the path of the Recycle Bin or other unique folder to the procedure.

```
Sub ShowFileSize(filespec)
   Dim fs, f, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set f = fs.GetFolder(filespec)
   s = UCase(f.Name) & " is a " & f.Type
   MsgBox s, 0, "File Size Info"
End Sub
```

TypeName Function

Description

Returns a string that provides **Variant** subtype information about a variable.

Syntax

TypeName(*varname***)**

The required *varname* argument can be any variable.

Return Values

The **TypeName** function has the following return values:

Value	Description
Byte	Byte value
Integer	Integer value
Long	Long integer value
Single	Single-precision floating-point value
Double	Double-precision floating-point value
Currency	Currency value
Decimal	Decimal value
Date	Date or time value
String	Character string value
Boolean	Boolean value; **True** or **False**
Empty	Unitialized
Null	No valid data
<object type>	Actual type name of an object
Object	Generic object
Unknown	Unknown object type
Nothing	Object variable that doesn't yet refer to an object instance
Error	Error

UBound Function

Description

Returns the largest available subscript for the indicated dimension of an array.

Syntax

UBound(*arrayname***[,** *dimension***])**

The **UBound** function syntax has these parts:

Part	Description
arrayname	Required. Name of the array variable; follows standard variable naming conventions.
dimension	Optional. Whole number indicating which dimension's upper bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If *dimension* is omitted, 1 is assumed.

Remarks

The **UBound** function is used with the **LBound** function to determine the size of an array. Use the **LBound** function to find the lower limit of an array dimension.

The default lower bound for any dimension is always 0. As a result, **UBound** returns the following values for an array with these dimensions:

```
Dim A(100,3,4)
```

Statement	Return Value

UBound(A, 1)	99
UBound(A, 2)	2
UBound(A, 3)	3

UCase Function

Description

Returns a string that has been converted to uppercase.

Syntax

UCase(*string***)**

The *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

Remarks

Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.

VarType Function

Description

Returns a value indicating the subtype of a variable.

Syntax

VarType(*varname***)**

The *varname* argument can be any variable.

Return Values

The **VarType** function returns the following values:

Constant	Value	Description
vbEmpty	0	**Empty** (uninitialized)
vbNull	1	**Null** (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency
vbDate	7	Date
vbString	8	String
vbObject	9	Automation object
vbError	10	Error
vbBoolean	11	Boolean
vbVariant	12	Variant (used only with arrays of Variants)
vbDataObject	13	A data-access object
vbByte	17	Byte
vbArray	8192	Array

Note These constants are specified by VBScript. As a result, the names can be used anywhere in your code in place of the actual values.

Remarks

The **VarType** function never returns the value for Array by itself. It is always added to some other value to indicate an array of a particular type. The value for Variant is only returned when it has been added to the value for Array to indicate that the argument to the **VarType** function is an array. For example, the value returned for an array of integers is calculated as 2 + 8192, or 8194. If an object has a default property, **VarType (***object***)** returns the type of its default property.

VolumeName Property

Description

Sets or returns the volume name of the specified drive. Read/write.

Syntax

object.**VolumeName** [= *newname*]

The **VolumeName** property has these parts:

Part	Description
object	Required. Always the name of a **Drive** object.
newname	Optional. If provided, *newname* is the new name of the specified *object*.

Remarks

The following code illustrates the use of the **VolumeName** property:

```
Sub ShowVolumeInfo(drvpath)
   Dim fs, d, s
   Set fs = CreateObject("Scripting.FileSystemObject")
   Set d = fs.GetDrive(fs.GetDriveName(fs.GetAbsolutePathName(drvpath)))
   s = "Drive " & d.DriveLetter & ": - " & d.VolumeName
   MsgBox s
End Sub
```

Weekday Function

Description

Returns a whole number representing the day of the week.

Syntax

Weekday(*date***,** [*firstdayofweek*]**)**

The **Weekday** function syntax has these arguments:

Part	Description
date	Any expression that can represent a date. If *date* contains **Null**, **Null** is returned.
firstdayofweek	A constant that specifies the first day of the week. If omitted, **vbSunday** is assumed.

Settings

The *firstdayofweek* argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Return Values

The **Weekday** function can return any of these values:

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

WeekdayName Function

Description

Returns a string indicating the specified day of the week.

Syntax

WeekdayName(*weekday***,** *abbreviate***,** *firstdayofweek***)**

The **WeekdayName** function syntax has these parts:

Part	Description
weekday	Required. The numeric designation for the day of the week. Numeric value of each day depends on setting of the *firstdayofweek* setting.
abbreviate	Optional. Boolean value that indicates if the weekday name is to be abbreviated. If omitted, the default is **False**, which means that the weekday name is not abbreviated.
firstdayofweek	Optional. Numeric value indicating the first day of the week. See Settings section for values.

Settings

The *firstdayofweek* argument can have the following values:

Constant	Value	Description
vbUseSystem	0	Use National Language Support (NLS) API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

While...Wend Statement

Description

Executes a series of statements as long as a given condition is **True**.

Syntax

While *condition*
[*statements*]
Wend

The **While...Wend** statement syntax has these parts:

Part	Description
condition	Numeric or string expression that evaluates to **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**.
statements	One or more statements executed while condition is **True**.

Remarks

If *condition* is **True**, all statements in *statements* are executed until the **Wend** statement is encountered. Control then returns to the **While** statement and *condition* is again checked. If *condition* is still **True**, the process is repeated. If it is not **True**, execution resumes with the statement following the **Wend** statement.

While...Wend loops can be nested to any level. Each **Wend** matches the most recent **While**.

Tip The **Do...Loop** statement provides a more structured and flexible way to perform looping.

Write Method

Description

Writes a specified string to a **TextStream** file.

Syntax

object.**Write(***string***)**

The **Write** method syntax has these parts:

Part	Description
object	Required. Always the name of a **TextStream** object.
string	Required. The text you want to write to the file.

Remarks

Specified strings are written to the file with no intervening spaces or characters between each string. Use the **WriteLine** method to write a newline character or a string that ends with a newline character.

WriteBlankLines Method

Description

Writes a specified number of newline characters to a **TextStream** file.

Syntax

object.**WriteBlankLines(***lines***)**

The **WriteBlankLines** method syntax has these parts:

Part	Description
object	Required. Always the name of a **TextStream** object.
lines	Required. Number of newline characters you want to write to the file.

WriteLine Method

Description

Writes a specified string and newline character to a **TextStream** file.

Syntax

object.**WriteLine**([*string*])

The **WriteLine** method syntax has these parts:

Part	Description
object	Required. Always the name of a **TextStream** object.
string	Optional. The text you want to write to the file. If omitted, a newline character is written to the file.

Xor Operator

Description

Used to perform a logical exclusion on two expressions.

Syntax

result **=** *expression1* **Xor** *expression2*

The **Xor** operator syntax has these parts:

Part	Description
result	Any numeric variable.
expression1	Any expression.
expression2	Any expression.

Remarks

If one, and only one, of the expressions evaluates to **True**, *result* is **True**. However, if either expression is **Null**, *result* is also **Null**. When neither expression is **Null**, *result* is determined according to the following table:

If *expression1* is	And *expression2* is	Then *result* is
True	**True**	**False**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

The **Xor** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	Then *result* is
0	0	0
0	1	1
1	0	1
1	1	0

Year Function

Description

Returns a whole number representing the year.

Syntax

Year(*date*)

The *date* argument is any expression that can represent a date. If *date* contains **Null**, **Null** is returned.

Color Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbBlack	&h00	Black
vbRed	&hFF	Red
vbGreen	&hFF00	Green
vbYellow	&hFFFF	Yellow
vbBlue	&hFF0000	Blue
vbMagenta	&hFF00FF	Magenta
vbCyan	&hFFFF00	Cyan
vbWhite	&hFFFFFF	White

Comparison Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Date/Time Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday
vbFirstJan1	1	Use the week in which January 1 occurs (default).
vbFirstFourDays	2	Use the first week that has at least four days in the new year.
vbFirstFullWeek	3	Use the first full week of the year.
vbUseSystem	0	Use the date format contained in the regional settings for your computer.
vbUseSystemDayOfWeek	0	Use the day of the week specified in your system settings for the first day of the week.

Date Format Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbGeneralDate	0	Display a date and/or time. For real numbers, display a data and time. If there is no fractional part, display only a date. If there is no integer part, display time only. Date and time display is determined by your system settings.
vbLongDate	1	Display a date using the long date format specified in your computer's regional settings.
vbShortDate	2	Display a date using the short date format specified in your computer's regional settings.
vbLongTime	3	Display a time using the long time format specified in your computer's regional settings.
vbShortTime	4	Display a time using the short time format specified in your computer's regional settings.

DriveType Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
Unknown	0	Drive type can't be determined.
Removable	1	Drive has removable media. This includes all floppy drives and many other varieties of storage devices.
Fixed	2	Drive has fixed (nonremovable) media. This includes all hard drives, including hard drives that are removable.
Remote	3	Network drives. This includes drives shared anywhere on a network.
CDROM	4	Drive is a CD-ROM. No distinction is made between read-only and read/write CD-ROM drives.
RAMDisk	5	Drive is a block of Random Access Memory (RAM) on the local computer that behaves like a disk drive.

File Attribute Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
Normal	0	Normal file. No attributes are set.
ReadOnly	1	Read-only file.
Hidden	2	Hidden file.
System	4	System file.
Volume	8	Disk drive volume label.
Directory	16	Folder or directory.
Archive	32	File has changed since last backup.
Alias	64	Link or shortcut.
Compressed	128	Compressed file.

File Input/Output Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing. If a file with the same name exists, its previous contents are overwritten.
ForAppending	8	Open a file and write to the end of the file.

Miscellaneous Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbObjectError	-2147221504	User-defined error numbers should be greater than this value, for example, Err.Raise Number = vbObjectError + 1000

MsgBox Constants

The following constants are used with the **MsgBox** function to identify what buttons and icons appear on a message box and which button is the default. In addition, the modality of the **MsgBox** can be specified. These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbOKOnly	0	Display **OK** button only.
vbOKCancel	1	Display **OK** and **Cancel** buttons.
vbAbortRetryIgnore	2	Display **Abort**, **Retry**, and **Ignore** buttons.
vbYesNoCancel	3	Display **Yes**, **No**, and **Cancel** buttons.
vbYesNo	4	Display **Yes** and **No** buttons.
vbRetryCancel	5	Display **Retry** and **Cancel** buttons.
vbCritical	16	Display **Critical Message** icon.
vbQuestion	32	Display **Warning Query** icon.
vbExclamation	48	Display **Warning Message** icon.
vbInformation	64	Display **Information Message** icon.
vbDefaultButton1	0	First button is the default.
vbDefaultButton2	256	Second button is the default.
vbDefaultButton3	512	Third button is the default.
vbDefaultButton4	768	Fourth button is the default.
vbApplicationModal	0	Application modal. The user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal. All applications are suspended until the user responds to the message box.

The following constants are used with the **MsgBox** function to identify which button a user has selected. These constants are only available when your project has an explicit reference to the appropriate type

library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbOK	1	**OK** button was clicked.
vbCancel	2	**Cancel** button was clicked.
vbAbort	3	**Abort** button was clicked.
vbRetry	4	**Retry** button was clicked.
vbIgnore	5	**Ignore** button was clicked.
vbYes	6	**Yes** button was clicked.
vbNo	7	**No** button was clicked.

SpecialFolder Constants
These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
WindowsFolder	0	The Windows folder contains files installed by the Windows operating system.
SystemFolder	1	The System folder contains libraries, fonts, and device drivers.
TemporaryFolder	2	The Temp folder is used to store temporary files. Its path is found in the TMP environment variable.

String Constants

The following string constants can be used anywhere in your code in place of actual values:

Constant	Value	Description
vbCr	**Chr**(13)	Carriage return
vbCrLf	**Chr**(13) **& Chr**(10)	Carriage return–linefeed combination
vbFormFeed	**Chr**(12)	Form feed; not useful in Microsoft Windows
vbLf	**Chr**(10)	Line feed
vbNewLine	**Chr**(13) **& Chr**(10) or **Chr**(10)	Platform-specific newline character; whatever is appropriate for the platform
vbNullChar	**Chr**(0)	Character having the value 0
vbNullString	String having value 0	Not the same as a zero-length string (""); used for calling external procedures
vbTab	**Chr**(9)	Horizontal tab
vbVerticalTab	**Chr**(11)	Vertical tab; not useful in Microsoft Windows

Tristate Constants

These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
TristateTrue	-1	True
TristateFalse	0	False
TristateUseDefault	-2	Use default setting

VarType Constants
These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

Constant	Value	Description
vbEmpty	0	Uninitialized (default)
vbNull	1	Contains no valid data
vbInteger	2	Integer subtype
vbLong	3	Long subtype
vbSingle	4	Single subtype
vbDouble	5	Double subtype
vbCurrency	6	Currency subtype
vbDate	7	Date subtype
vbString	8	String subtype
vbObject	9	Object
vbError	10	Error subtype
vbBoolean	11	Boolean subtype
vbVariant	12	Variant (used only for arrays of variants)
vbDataObject	13	Data access object
vbDecimal	14	Decimal subtype
vbByte	17	Byte subtype
vbArray	8192	Array