**VB BUILT-IN FUNCTIONS AND STATEMENTS**

**Nanosoft**

**MTM SUHAIL  
MUTHIRATHODIKA  
POOKKOTTUMPADAM (PO)   
MALAPPURAM  
KERALA  
PIN: 679332**

**PH: 9047717702**

About…

Visual Basic provides more than 200 built-in statements and functions. Some are used frequently; some are seldom used at all. Even experienced programmers sometimes happen upon a statement or function they didn't know about. This appendix can be considered a quick-reference guide to all of VB's built-in statements and functions. It offers the syntax for each, as well as a brief description of what it does and how its arguments (if any) are used. This is by no means a complete reference. Microsoft's Visual Basic 6.0 Language Reference is probably the best source for in-depth information on VB's statements and functions. If you have any questions about syntax or use of a particular statement or function, then you are strongly urged to consult that reference to obtain additional information.

# Functions and Statements

**Abs Function**

**Abs (number)**

##### Returns the absolute value of number. The data type returned is the same as the data type of the number argument.

**AppActivate Statement**

**AppActivate title [, wait]**

##### Activates the application window that has the string title in its title bar or, alternatively, the task ID specified by title. The optional wait argument (Boolean) can be used to specify whether the calling application should wait until it has the focus before the application window is activated.

**Array Function**

**Array (arglist)**

##### Returns a Variant data item that contains an array. arglist refers to a comma-delimited list of values that make up the elements of the array, with the first value corresponding to the first element of the array, the second value corresponding to the second element of the array, and so on.

**Asc Function**

**Asc (string)**

##### Returns an Integer value that represents the ASCII code for the first character in the string.

**Atn Function**

**Atn(number)**

##### Returns a Double value that is the arctangent of number.

**Beep Statement**

**Beep**

##### Sounds a tone through the PC's speaker. Frequency and duration of the tone may vary from system to system.

**Call Statement**

**[Call] name [argument list]**

##### Executes a sub, function, or DLL procedure. The name argument specifies the name of the procedure to call, and argument list is an optional list of arguments that will be passed to the called procedure. The Call keyword is optional, but if it is included, then at least one or more arguments for argumentlist must also be included.

**CBool Function**

**CBool(expression)**

##### Converts the value of expression to a Boolean data type. The argument expression can be any valid string or numeric expression.

**CByte Function**

**CByte(expression)**

##### Converts the value of expression to a Byte data type. The argument expression must be a numeric value between 0 and 255.

**CCur Function**

**CCur(expression)**

##### Converts the value of expression to a Currency data type. The argument expression must be a numeric value between -922,337,203,685,477.5808 and 922,337,203,685,477.5807.

**CDate Function**

**CDate(expression)**

##### Converts the value of expression to a Date data type. The argument expression must be a valid date expression.

**CDbl Function**

**CDbl(expression)**

##### Converts the value of expression to a Double data type. The argument expression must be a numeric value between -1.79769313486232E308 and -4.94065645841247E-324 for negative values, or between 4.94065645841247E-324 and 1.79769313486232E308 for positive values.

**CDec Function**

**CDec(expression)**

##### Converts the value of expression to a Decimal data type. The argument expression must be a numeric value of +/-79,228,162,514,264,337,593,543,950,335 for zero-scaled numbers (numbers with no decimal places), or +/-7.9228162514264337593543950335 for numbers with 28 decimal places.

**ChDir Statement**

**ChDir path**

##### Changes the current directory to the one specified by the path argument. Note that although ChDir changes the default directory, it does not change the default drive as well.

**ChDrive Statement**

**ChDrive drive**

##### Changes the current drive to the one specified by the drive argument.

**Choose Function**

**Choose(index, choice-1[, choice-2, ... [, choice-n]])**

##### Returns a value from a list of choices (specified by the arguments choice-1 through choice-n) based on the value of the index argument. If index is 1, then the value returned by the Choose function will be the value represented by choice-1; if index is 2, then the value returned will be that of choice-2; and so on.

**Chr Function**

**Chr(charcode)**

##### Returns a one-character String value that represents the ASCII character of the number specified by the charcode argument.

**CInt Function**

**CInt(expression)**

##### Converts the value of expression to an Integer data type. The argument expression must be a numeric value from -32,768 to 32,767. Fractions are rounded.

**CLng Function**

**CLng(expression)**

##### Converts the value of expression to a Long data type. The argument expression must be a numeric value from -2,147,483,648 to 2,147,483,647. Fractions are rounded.

**Close Statement**

**Close [filenumberlist]**

##### Closes any files opened with the Open statement that corre- spond to the file numbers specified by filenumberlist. The filenumberlist argument can contain a single file number (for example, #1) or multiple file numbers (for example, #1, #4, #5). If the filenumberlist argument is omitted, all open files will be closed.

**Command Function**

**Command**

##### Returns any command-line arguments specified when launching Visual Basic. For compiled programs, Command returns the command-line arguments specified when the program was launched.

**Const Statement**

**[Public | Private] Const constname [As type] = expression**

##### Declares a constant with the name constname and the value represented by expression. The Public and Private keywords define the constant's scope, and the optional As type specifies the constant's data type. If the as type is omitted, the constant will be declared as the data type most appropriate for expression.

**Cos Function**

**Cos(number)**

##### Returns a Double value that is the cosine of the angle specified by the number argument.

**CreateObject Function**

**CreateObject (class)**

##### Creates and returns a reference to an ActiveX object of type class. The class argument should use the syntax appname.objecttype, where appname is the name of the application providing the object, and objecttype is the class of the object to be created.

**CSng Function**

**CSng(expression)**

##### Converts the value of expression to a Single data type. The argument expression must be a numeric value from -3.402823E38 to -1.401298E-45 for negative values, or from 1.401298E-45 to 3.402823E38 for positive values.

**CStr Function**

**CStr(expression)**

##### Converts the value of expression to a String data type. The string that CStr returns depends on the data type of the expression argument: For Booleans, CStr returns either True or False; for Dates, CStr returns a date based on the short date format on your system; for Errors, CStr returns the word Error followed by the error number. expression values that are Empty return a zero-length string, and numeric expression values return a string containing the number. A Null expression value will cause a runtime error.

**CurDir Function**

**CurDir[(drive)]**

##### Returns a String value that represents the full path of the current directory. If the drive argument is given, then CurDir returns the directory path of the current directory for that drive.

**CVar Function**

**CVar(expression)**

##### Converts the value of expression to a Variant data type. The argument expression can be either numeric or a string.

**CVErr Function**

**CVErr(errornumber)**

##### Returns a Variant of subtype Error that contains the error number specified by the errornumber argument.

**Date Function**

**Date**

##### Returns a Variant of subtype Date that contains the current system date.

**Date Statement**

**Date = date**

##### Sets the current system date as specified by the date argument. For Windows 95 systems, date must be a valid date between January 1, 1980 and December 31, 2099. For Windows NT systems, date must be a valid date between January 1, 1980 and December 31, 2079.

**DateAdd Function**

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

##### Returns a Variant of subtype Date calculated by taking the date specified by the date argument and adding or subtracting the amount of time specified by interval and time. The interval argument contains a code that represents a unit of time (for example, yyyy for years, m for months, d for days), and number is the number of units to be added to date.

**DateDiff Function**

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

##### Returns a Variant of subtype Long that represents the number of time units between two dates (date1 and date2). The interval argument contains a code that represents the unit of time (for example, yyyy for years) that will be returned by the function. The optional firstdayofweek and firstweekofyear arguments are used to specify how the time difference should be calculated when certain codes are used for interval.

**DatePart Function**

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

##### Returns a Variant of subtype Integer that contains the part of date specified by interval. The interval argument contains a code that represents the unit of time (for example, ww for weeks) that will be returned by the function. The optional firstdayofweek and firstweekofyear arguments are used to specify how the date should be calculated when certain interval codes are used.

**DateSerial Function**

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

##### Returns a Variant of subtype Date that represents a date as specified by the year, month, and day arguments.

**DateValue Function**

**DateValue(date)**

##### Returns a Variant of subtype Date that is derived from the date value specified by the date argument.

**Day Function**

**Day(date)**

##### Returns a Variant of subtype Integer that represents the day of the month (1[nd]31) for the date value specified by the date argument.

**DDB Function**

**DDB(cost, salvage, life, period[, factor])**

##### Returns a Double value that represents the depreciation of an asset for a specified amount of time using a given method of depreciation. The cost argument represents the initial cost of the asset, salvage represents the value of the asset at the end of its working lifetime, life represents the lifetime of the asset, and period represents the period (in months) for which the depreciation is calculated. The optional factor argument specifies the rate at which the balance declines. If it is omitted, then the double-declining depreciation method is used.

**Declare Statement**

**[Public | Private] Declare Sub name Lib "libname" [Alias \_ "aliasname"][([arglist])]**

or

**[Public | Private] Declare Function name Lib "libname"**

**[Alias "aliasname"][([arglist])][As type]**

##### Declares references to Sub or Function procedures in an external DLL (dynamic-link library). The optional Public and Private keywords define the procedure's scope. The name argument is the name of the procedure, and the libname argument specifies the DLL that contains the procedure. The optional aliasname argument specifies an alternate name for the procedure in the DLL. arglist is a list of arguments passed to the procedure. For Function procedures, the As type specifies the data type of the value returned by the Function. Declare statements can only be used at module level.

**DefBool Statement**

**DefBool letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Boolean data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefBool can only be used at module level.

**DefByte Statement**

**DefByte letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Byte data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefByte can only be used at module level.

**DefCur Statement**

**DefCur letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Currency data type. The letterrange arguments should be constructed as letter1 [-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefCur can only be used at module level.

**DefDate Statement**

**DefDate letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Date data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefDate can only be used at module level.

**DefDbl Statement**

**DefDbl letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Double data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefDbl can only be used at module level.

**DefDec Statement**

**DefDec letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Decimal data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefDec can only be used at module level.

**DefInt Statement**

**DefInt letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Integer data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefInt can only be used at module level.

**DefLng Statement**

**DefLng letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Long data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefLng can only be used at module level.

**DefObj Statement**

**DefObj letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Object data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefObj can only be used at module level.

**DefSng Statement**

**DefSng letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Single data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefSng can only be used at module level.

**DefStr Statement**

**DefStr letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the String data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefStr can only be used at module level.

**DefVar Statement**

**DefVar letterrange[, letterrange]...**

##### Specifies that all variables and function return values that begin with the letters specified by the letterrange arguments are automatically defined to be of the Variant data type. The letterrange arguments should be constructed as letter1[-letter2], where letter1 is the first (or only) letter in the range, and letter2 is the last letter in the range. DefVar can only be used at module level.

**DeleteSetting Statement**

**DeleteSetting appname, section[, key]**

##### Deletes an application's section or key setting entries from the System Registry. The appname argument specifies the name of the application, and section is the name of the section to be deleted. If the optional key argument is used, then only that key (and not the whole section) will be deleted.

**Dim Statement**

**Dim [WithEvents] varname[([subscripts])] [As [New] type] [,[WithEvents]**

**varname[([subscripts])] [As [New] type]]...**

##### Declares one or more variables or objects. The varname argument is the name of the variable, and the optional As [New] type indicates its data type. If the New keyword is used, then an implicit creation of the object is made. The optional WithEvents keyword (valid only when the Dim statement is used in class modules) indicates that varname is an object variable as is used to respond to events triggered by an ActiveX object. The optional subscripts are the dimensions of an array variable.

**Dir Function**

**Dir[(pathname[, attributes])]**

##### Returns a String value containing the name of a file, directory, or folder that matches a pattern (specified in the pathname argument) and/or a file attribute (specified in attributes). The first time the Dir function is called, it returns the name of a file based on the pathname and attributes arguments. If the function is called again and no arguments are given, then it returns the second file name for the given pathname and attributes, and so on.

**Do...Loop Statement**

**Do [{While | Until} condition]**

**[statements]**

**[Exit Do]**

**[statements]**

**Loop**

or

**Do**

**[statements]**

**[Exit Do]**

**[statements]**

**Loop [{While | Until} condition]**

##### Repeats one or more statements while a condition is True or until a condition becomes True. The optional Exit Do keywords pass control to the line of code immediately following the Do...Loop structure.

**DoEvents Function**

**DoEvents( )**

##### Temporarily gives control to the operating system so that it can process other events. The DoEvents function is typically used inside loops so that a program does not tie up system resources for a long period of time.

**End Statement**

**End**

**End Function**

**End If**

**End Property**

**End Select**

**End Sub**

**End Type**

**End With**

##### Ends a program (End), procedure (End Function, End Property, or End Sub), type structure (End Type), or program block (End If, End Select, or End With).

**Enum Statement**

**[Public | Private] Enum name**

**membername [= constantexpression]**

**membername [= constantexpression]**

**...**

**End Enum**

##### Declares an enumeration type named name that is composed of one or more members specified by membername. Members can be assigned values using constantexpression. The optional Public and Private keywords define the enumeration's scope.

**Environ Function**

**Environ({envstring | number})**

##### Returns the String value of the operating system variable specified by envstring or, alternatively, the numeric order of the environment string in the environment-string table specified by number.

**EOF Function**

**EOF(filenumber)**

##### Returns a True or False value (Integer) that indicates whether the end of file marker has been reached for the Random or Input file associated with the filenumber argument.

**Erase Statement**

**Erase arraylist**

##### Reinitializes the elements in one or more fixed-size array and frees up the dynamic-array storage space that was taken up by the array(s). The arraylist argument is one or more comma-delimited array names.

**Error Function**

**Error[(errornumber)]**

##### Returns a String value that contains the error message that corresponds to the errornumber argument.

**Error Statement**

**Error errornumber**

##### Causes an error to occur. The errornumber argument indicates the type of error that should occur.

**Event Statement**

**[Public] Event procedurename [(arglist)]**

##### Declares a user-defined event with the name procedurename. The optional Public keyword indicates that the Event should be visible throughout the project, even though that is the default. The optional argument list (arglist) should contain one or more arguments defined using the syntax:

**[ByVal | ByRef] varname[()] [As type]**

##### where varname is the name of the argument, As type indicates the data type of the argument, and the optional ByRef or ByVal keywords specify whether the argument should be passed by reference (ByRef) or by value (ByVal). If ByRef and ByVal are not specified, then the argument will be passed by reference.

**Exit Statement**

**Exit Do**

**Exit For**

**Exit Function**

**Exit Property**

**Exit Sub**

##### Exits a procedure (Exit Function, Exit Property, or Exit Sub) or looping structure (Exit Do or Exit For).

**Exp Function**

**Exp(number)**

##### Returns a Double value that is e (the base of natural logarithms) raised to the power specified by number.

**FileAttr Function**

**FileAttr(filenumber, returntype)**

##### Returns a Long value that indicates the file mode for a file opened using the Open statement. The argument filenumber is the file number for the open file, and returntype indicates the type of information to be returned. Although returntype can be set to 2 to return the operating system file handle for the open file, it only works on 16-bit systems and should be avoided in VB5. Instead, use a value of 1 for returntype to return the open file type. The possible values returned by the FileAttr function for indicating file type are: 1 for Input, 2 for Output, 4 for Random, 8 for Append, and 32 for Binary.

**FileCopy Statement**

**FileCopy source, destination**

##### Copies the filename and path specified by the source argument to the filename and path specified by the destination argument.

**FileDateTime Function**

**FileDateTime(pathname)**

##### Returns a Variant of subtype Date that indicates the date and time when the file specified by the pathname argument was last modified.

**FileLen Function**

**FileLen(pathname)**

##### Returns a Long value that contains the file size (in bytes) of the file specified by the pathname argument.

**Fix Function**

**Fix(number)**

##### Returns the integer portion of the number specified by the number argument. If number is negative, then Fix returns the first negative integer greater than or equal to number.

**For Each...Next Statement**

**For Each element In group**

**[statements]**

**[Exit For]**

**[statements]**

**Next [element]**

##### Executes one or more statements for each element in the array or collection specified by group. The optional Exit For can be used to immediately exit the looping structure.

**For...Next Statement**

**For counter = start To end [Step step]**

**[statements]**

**[Exit For]**

**[statements]**

**Next [counter]**

##### Executes one or more statements a specified number of times. The counter argument is a variable used to increment from start to end. By default, counter is incremented by 1 each time the loop is executed, although the optional step argument can be used to specify a different increment. The optional Exit For can be used to immediately exit the looping structure.

**Format Function**

**Format(expression[, format[, firstdayofweek[, \_ firstweekofyear]]])**

##### Returns a Variant of subtype String that contains the value specified by expression using a format defined by the format argument. The format argument uses codes (for example, d for days or # for numbers) to determine how expression will be formatted. The optional firstdayofweek and firstweekofyear arguments are used when formatting certain values.

**FreeFile Function**

**FreeFile[(rangenumber)]**

##### Returns an Integer value the represents the next file number available for use with the Open statement. The optional rangenumber argument can be used to specify which range of file numbers should be used: 0 (the default) for file numbers in the range of 1-255, or 1 for file numbers in the range of 256-511.

**Function Statement**

**[Public | Private | Friend] [Static] Function name \_ [(arglist)] [As type]**

**[statements]**

**[name = expression]**

**[Exit Function]**

**[statements]**

**[name = expression]**

**End Function**

##### Declares the various parts of a Function procedure. The optional Public, Private, and Friend keywords can be used to define the Function's scope, and the optional Static keyword indicates that the procedure's local variables are preserved between calls to the Function. The name argument specifies the name of the Function procedure and can be assigned a value (name = expression) that will be returned by the procedure. The data type of the return value can be specified using the As type clause. The optional Exit Function can be used to exit the Function procedure immediately.

##### The optional list of arguments (arglist) defines the arguments that will be passed to the procedure. The arguments should use the following syntax:

##### **[Optional] [ByVal | ByRef] [ParamArray] varname[()] [As \_ type] [= default value]**

##### where the Optional keyword can be used to specify that the argument is not required (default value assigns the argument's default value), ByVal and ByRef determine whether the argument should be passed by value or by reference (the default), and the ParamArray keyword specifies that the argument is an Optional array of Variant elements. ParamArray can only be used with the last argument in the argument list.

**FV Function**

**FV(rate, nper, pmt[, pv[, type]])**

##### Returns a Double value that indicates the future value of an annuity based on a number (nper) of periodic fixed payment amounts (pmt) and a fixed interest rate (rate). The optional pv argument specifies a present value or lump sum of a series of future payments, and the optional type argument specifies when payments are due (0 for end of the payment period, the default; 1 for beginning of the payment period).

**Get Statement**

**Get [#]filenumber,[ recnumber,] varname**

##### Reads data from the open disk file corresponding to the filenumber argument into a variable (varname). Get works with files open as Random or Binary, and a record number (recnumber) can be specified when retrieving data from a Random file. When using Binary files, recnumber can alternatively be used to specify the byte position from which the data is to be read.

**GetAllSettings Function**

**GetAllSettings(appname, section)**

##### Returns a list of key settings and their values from a specific application (appname) entry and section (section) in the System Registry.

**GetAttr Function**

**GetAttr(pathname)**

##### Returns an Integer value that represents the attributes for the file, directory, or folder specified by the pathname argument. The value returned can be compared bitwise with several VB constants (vbNormal, vbReadOnly, vbHidden, vbSystem, vbDirectory, and vbArchive) to determine which attributes are set.

**GetAutoServerSettings Function**

**object.GetAutoServerSettings([progid], [clsid])**

##### Returns a Variant array that contains information concerning the state of an ActiveX object's registration. The optional progid and clsid can be included to specify the object's ProgID and CLSID, respectively. The values of the elements of the Variant array that are returned by GetAutoServerSettings are, in order: local/remote registration of object (True if the object is registered remotely), the remote machine name, the RPC network protocol name, and the RPC authentication level.

**GetObject Function**

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

##### Returns a reference to an object of type class. The pathname argument can be included to specify the path and filename from which the object should be retrieved; however, if it is omitted, then the class name must be specified.

**GetSetting Function**

**GetSetting(appname, section, key[, default])**

##### Returns a single key setting value from a specific application (appname) entry and section (section) in the System Registry. If no value is set for the key setting specified, then the optional default value can be returned. If default is omitted, then the default value returned will be a zero-length string.

**GoSub...Return Statement**

**GoSub line**

**...**

**line**

**...**

**Return**

##### Transfers program control to the subroutine indicated by the line label or line number line until the Return statement is reached; then returns control to the line of code immediately following the GoSub statement.

**GoTo Statement**

**GoTo line**

##### Transfers program control to the line of code specified by the line label or line number line. The line must be inside the procedure that contains the GoTo statement.

**Hex Function**

**Hex(number)**

##### Returns a String value that represents the hexadecimal value of the argument number.

**Hour Function**

**Hour(time)**

##### Returns a Variant of subtype Integer that represents the hour (0[nd]23) of the time value specified by the time argument.

**If...Then...Else Statement**

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

or

**If condition Then**

**[statements]**

**[ElseIf condition-n Then**

**[elseifstatements] ...**

**[Else**

**[elsestatements]]**

**End If**

##### Conditionally executes one or more statements if the value expressed by condition is True. One or more ElseIf clauses can be included to test other conditions and execute other statements (elseifstatements) if the preceding condition is False. An Else clause can also be included to execute other statements (elsestatements) if none of the preceding conditions are True.

**IIf Function**

**IIf(expression, truepart, falsepart)**

##### Returns one of two values based on whether expression evaluates to True or False. If True, then IIf returns the truepart value; if False, then the falsepart value is returned.

**IMEStatus Function**

**IMEStatus**

##### Returns an Integer value that represents Windows' current Input Method Editor (IME) mode. IMEStatus is only available in Far East versions.

**Implements Statement**

**Implements [interfacename | class]**

##### Specifies an interface (interfacename) or class (class) that will be implemented in the class module in which the Implements statement is used.

**Input # Statement**

**Input #filenumber, varlist**

##### Reads data from the open file associated with the filenumber argument and places it in the variables in the varlist argument. The varlist argument should contain one or more comma-delimited variables.

**Input Function**

**Input(number, [#]filenumber)**

##### Returns a String value containing characters read in from the open file that corresponds to the filenumber argument. The number of characters to be read in are specified by the number argument.

**InputBox Function**

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

##### Displays a dialog box and waits for the user to enter text or click a button; then returns what the user entered in a String value. The prompt argument specifies the message to be displayed in the dialog box, title specifies an optional caption for the dialog box's title bar, and default specifies the optional default value returned by the InputBox function if no value is entered by the user. The optional xpos and ypos arguments specify (in twips) the horizontal and vertical position of the dialog box on the screen. The optional helpfile and context arguments are used to provide context-sensitive Help for the dialog box.

**InStr Function**

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

##### Returns a Variant of subtype Long that specifies the starting position of the first occurrence of a substring (string2) within another string (string1). The optional start argument specifies from which character in string2 to start searching; the default is 1 (first character). The optional compare argument specifies the type of string comparison that will be made (0 for binary or 1 for textual noncase-sensitive).

**Int Function**

**Int(number)**

##### Returns the integer portion of the number specified by the number argument. If number is negative, then Int returns the first negative integer less than or equal to number.

**IPmt Function**

**IPmt(rate, per, nper, pv[, fv[, type]])**

##### Returns a Double value that indicates the interest payment for a fixed-period annuity based on a number (nper) of periodic fixed payments (per) and a fixed interest rate (rate). The pv argument specifies the present value of a series of payments or receipts. The optional fv argument specifies the future value or cash balance left after the final payment. The optional type argument specifies when payments are due (0 for end of the payment period, the default; 1 for beginning of the payment period).

**IRR Function**

**IRR(values()[, guess])**

##### Returns a Double value indicating the internal rate of return for an array of values that represent cash flow. The values() array must contain at least one negative value (payment) and one positive value (receipt). The optional guess argument specifies an estimate value to be returned by IRR (default estimate is .1).

**IsArray Function**

**IsArray(varname)**

##### Returns a Boolean value that indicates whether the variable specified by varname is an array.

**IsDate Function**

**IsDate(expression)**

##### Returns a Boolean value that indicates whether expression is capable of being converted to a date value.

**IsEmpty Function**

**IsEmpty(expression)**

##### Returns a Boolean value that indicates whether a numeric or string expression has been initialized.

**IsError Function**

**IsError(expression)**

##### Returns a Boolean value that indicates whether a given expression is an error value.

**IsMissing Function**

**IsMissing(argname)**

##### Returns a Boolean value that indicates whether an optional Variant argument (argname) has been passed to a procedure. IsMissing returns True if no value has been provided for the specified argument.

**IsNull Function**

**IsNull(expression)**

##### Returns a Boolean value that indicates whether a given expression contains no data and is Null.

**IsNumeric Function**

**IsNumeric(expression)**

##### Returns a Boolean value that indicates whether a given expression can be evaluated as a numeric value.

**IsObject Function**

**IsObject(identifier)**

##### Returns a Boolean value that indicates whether a given identifier represents an object variable.

**Kill Statement**

**Kill pathname**

##### Deletes the file(s) or directory represented by the pathname argument. Filenames in the pathname argument can contain wildcards, allowing multiple files to be deleted.

**LBound Function**

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

##### Returns a Long value that represents the smallest subscript for a dimensioned array (arrayname). For multidimensional arrays, the dimension argument can be included to specify which dimension should be used.

**LCase Function**

**LCase(string)**

##### Converts a string to all lowercase characters and returns a new String value.

**Left Function**

**Left(string, length)**

##### Returns a String value length characters long that is taken from the left side of a given string.

**Len Function**

**Len(string | varname)**

##### Returns a Long value that indicates the number of characters in a string or, alternatively, the number of bytes required to store a particular variable (varname).

**Let Statement**

**[Let] varname = expression**

##### Assigns the value of an expression to a variable (varname). The Let keyword is usually omitted and is assumed by Visual Basic.

**Line Input # Statement**

**Line Input #filenumber, varname**

##### Reads a line of data (ending with a carriage return or carriage return-linefeed) from an open disk file corresponding to the filenumber argument. The data is placed in the String or Variant variable specified by varname.

**Load Statement**

**Load object**

##### Loads an object, such as a form or control, into memory.

**LoadPicture Function**

**LoadPicture([stringexpression])**

##### Loads the image specified by the stringexpression argument and returns it. This allows pictures to be loaded in and assigned to a Form's Picture property, a PictureBox control, or an Image control. If no stringexpression argument is given, then LoadPicture returns an empty picture.

**LoadResData Function**

**LoadResData(index, format)**

##### Loads data from the resource (.RES) file with the identifier of the index argument. The format argument specifies the format of the data (1 for cursors, 2 for bitmaps, 3 for icons, 4 for menus, 5 for dialog boxes, 6 for strings, 7 for font directories, 8 for fonts, 9 for accelerator tables, 10 for user-defined resources, 12 for group cursors, and 14 for group icons). The data returned by the LoadResData function can be assigned to a variable or object of the appropriate type.

**LoadResPicture Function**

**LoadResPicture(index, format)**

##### Loads a bitmap, icon, or cursor from the resource (.RES) file with the identifier of the index argument. The format argument specifies the format of the data (0 for bitmaps, 1 for icons, and 2 for cursors). The data returned by the LoadResPicture function can be assigned to an object of the appropriate type.

**LoadResString Function**

**LoadResString(index)**

##### Loads a string from the resource (.RES) file with the identifier of the index argument. The string that is returned can be assigned to a variable of String or Variant data type.

**Loc Function**

**Loc(filenumber)**

##### Returns a Long value that indicates the current byte position within the open file that corresponds to the filenumber argument.

**Lock Statement**

**Lock [#]filenumber[, recordrange]**

##### Prevents another process from accessing all or part of the open file that corresponds to the filenumber argument. The recordrange argument refers to a range of records (or bytes) that are to be locked and should use the syntax:

**recnumber | [start] To end**

##### where recnumber is the record number (for Random files) or byte position (for Binary files) where locking should begin. Alternatively, the starting and ending record numbers or bytes to be locked can be specified using the start and end arguments.

**LOF Function**

**LOF(filenumber)**

##### Returns a Boolean value that represents the byte size of the open file that corresponds to the filenumber argument.

**Log Function**

**Log(number)**

##### Returns a Double value that represents the natural logarithm of a specified number.

**LSet Statement**

**LSet stringvar = string**

or

**LSet varname1 = varname2**

##### In the first syntax, LSet assigns a string value to a String variable (stringvar), left-aligning the string to the String variable. In the second syntax, LSet copies a variable (varname2) from one user-defined type to a variable (varname1) in another user-defined type.

**LTrim Function**

**LTrim(string)**

##### Returns a Variant of subtype String that contains a copy of a given string with any leading spaces removed.

**Mid Function**

**Mid(string, start[, length])**

##### Returns a String value of one or more characters, taken from the String variable specified by the string argument. The start argument specifies the character position within string where the new String is to be obtained, and the optional length argument specifies how many characters are to be taken from string. If no length is specified, then all the characters in string (starting at the position given in the start argument) are used.

**Mid Statement**

**Mid(stringvar, start[, length]) = string**

##### Replaces one or more characters in a String variable (stringvar) with another string. The start argument specifies the character position within stringvar to place the new string, and the optional length argument specifies how many characters of string should be used. If length is omitted, then the entire string is used.

**Minute Function**

**Minute(time)**

##### Returns a Variant of subtype Integer that represents the minute (0-59) of the time value specified by the time argument.

**MIRR Function**

**MIRR(values(), financerate, reinvestrate)**

##### Returns a Double value that represents the modified internal rate of return for an array of values that represent cash flow. The values() array must contain at least one negative value (payment) and one positive value (receipt). The financerate argument specifies the interest rate paid as a cost of financing, and the reinvestrate argument specifies the interest rate received on gains from cash reinvestment.

**MkDir Statement**

**MkDir path**

##### Creates the new directory or folder specified by the path argument.

**Month Function**

**Month(date)**

##### Returns a Variant of subtype Integer that represents the month (1[nd]12) for the date value specified by the date argument.

**MsgBox Function**

**MsgBox(prompt[, buttons][, title][, helpfile, context]**

##### Displays a message in a dialog box with one or more buttons and waits for the user to respond. MsgBox then returns an Integer value that represents the button that was clicked. The prompt argument specifies the message to be displayed in the dialog box, and title specifies an optional caption for the dialog box's title bar. The optional buttons argument specifies which buttons will be displayed. The optional helpfile and context arguments are used to provide context-sensitive Help for the dialog box.

**Name Statement**

**Name oldpathname As newpathname**

##### Renames the file, directory, or folder specified by the oldpathname argument to the name specified by newpathname.

**Now Function**

**Now**

##### Returns a Variant of subtype Date that contains the current system date and time.

**NPer Function**

**NPer(rate, pmt, pv[, fv[, type]])**

##### Returns a Double value that indicates the number of periods for an annuity based on periodic fixed payments (pmt) and a fixed interest rate (rate). The pv argument specifies the present value of a series of payments or receipts. The optional fv argument specifies the future value or cash balance left after the final payment. The optional type argument specifies when payments are due (0 for end of the payment period, the default; 1 for beginning of the payment period).

**NPV Function**

**NPV(rate, values())**

##### Returns a Double value that represents the present value of an investment based on a discount rate (rate) and an array of values that represent cash flow. The values() array must contain at least one negative value (payment) and one positive value (receipt).

**Oct Function**

**Oct(number)**

##### Returns a String value that represents the octal value of the argument number.

**On Error Statement**

**On Error GoTo line**

**On Error Resume Next**

**On Error GoTo 0**

##### Enables or disables the use of an error-handling routine. The On Error statement can specify a line label or line number (specified by the line argument) to branch to when an error occurs, allowing error-handling to be enabled. Alternatively, using On Error Resume Next causes program control to be transferred to the line of code immediately following the line of code that causes an error. Finally, On Error GoTo 0 disables all error-handling.

**On...GoSub Statement**

**On expression GoSub destinationlist**

##### Evaluates a given expression and, depending on its value, transfers program control to a certain subroutine. The possible subroutines are contained in the destinationlist argument, which contains one or more comma-delimited line labels or line numbers. If expression evaluates to 1, then the first subroutine in the destinationlist is used; if it evaluates to 2, then the second subroutine in the destinationlist is used; and so on. Control is transferred to the line of code immediately following the On...GoSub line when a Return statement is encountered.

**On...GoTo Statement**

**On expression GoTo destinationlist**

##### Evaluates a given expression and, depending on its value, transfers program control to a certain line label or line number. The possible transfer points are contained in the destinationlist argument, which contains one or more comma-delimited line labels or line numbers. If expression evaluates to 1, then the first line label in the destinationlist is used; if it evaluates to 2, then the second line label in the destinationlist is used; and so on.

**Open Statement**

**Open pathname For mode [Access access] [lock] As \_ [#]filenumber [Len=reclength]**

##### Opens a file for input/output and assigns it to the given filenumber. The pathname argument specifies the name of the file to open, and mode indicates the file mode (Append, Binary, Input, Output, or Random). The optional Access clause can be used to specify permissions for the file (Read, Write, or Read Write). The optional lock argument can specify the operations that can be performed on the file by other processes (Shared, Lock Read, Lock Write, or Lock Read Write). The reclength argument can be used to specify the record size for random files or the buffer size for sequential files.

**Option Base Statement**

**Option Base [0 | 1]**

##### Declares the default lower bound of array subscripts. Option Base can only be used at module level.

**Option Compare Statement**

**Option Compare [Binary | Text | Database]**

##### Declares the default method used for string comparisons. Option Compare can only be used at module level.

**Option Explicit Statement**

**Option Explicit**

##### Forces explicit declaration of all variables in a module. If Option Explicit is not used, undeclared variables are automatically typed as Variants. Option Explicit can only be used at module level.

**Option Private Statement**

**Option Private Module**

##### Prevents a module's contents (that is, variables and objects) from being used outside its project. Option Private is only necessary when working with host applications that allow variables and objects to be referenced across multiple projects.

**Partition Function**

**Partition(number, start, stop, interval)**

##### Returns a Variant of subtype String that describes a range of numbers in which the number argument falls. The start and stop arguments specify the overall range of numbers, which is split up into smaller ranges as specified by the interval argument. The Partition function returns a string representation of the smaller range in which the number can be found, such as " 1: 10" for a number that falls in the range of 1 to 10.

**Pmt Function**

**Pmt(rate, nper, pv[, fv[, type]])**

##### Returns a Double value that indicates the payment for an annuity based on a number (nper) of periodic fixed payments and a fixed interest rate (rate). The pv argument specifies the present value of a series of payments or receipts. The optional fv argument specifies the future value or cash balance left after the final payment. The optional type argument specifies when payments are due (0 for end of the payment period, the default; 1 for beginning of the payment period).

**PPmt Function**

**PPmt(rate, per, nper, pv[, fv[, type]])**

##### Returns a Double value that indicates the principle payment for a given period (per) of an annuity based on a number (nper) of periodic fixed payments and a fixed interest rate (rate). The pv argument specifies the present value of a series of payments or receipts. The optional fv argument specifies the future value or cash balance left after the final payment. The optional type argument specifies when payments are due (0 for end of the payment period, the default; 1 for beginning of the payment period).

**Print # Statement**

**Print #filenumber, [outputlist]**

##### Writes data to the open sequential file that corresponds to filenumber. The optional outputlist argument can consist of one or more comma-delimited expressions to be written and should use the following syntax:

**[{Spc(n) | Tab[(n)]}] [expression][charpos]**

##### where Spc is optionally used to write n spaces, and Tab is optionally used to advance to the nth column number. The expression argument can specify the data to be written, and the charpos argument can specify the insertion point for the next character. If charpos is omitted, the next character will be written on the next line. If it is a semicolon, the next character will be written immediately following the last character.

**Private Statement**

**Private [WithEvents] varname[([subscripts])] [As [New] type][,[WithEvents]**

**varname[([subscripts])] [As [New] type]]...**

##### Declares one or more private variables. The varname argument specifies the name of the variable being declared, and subscripts are the dimensions for an array variable. The optional As [New] type clause can be used to specify the variable's data type, with the New keyword enabling implicit creation of an object. The optional WithEvents keyword specifies that the variable being declared is an object variable used to respond to events triggered by an ActiveX object. The Private statement can only be used at module level, and variables declared with it cannot be used outside their own module.

**Property Get Statement**

**[Public | Private | Friend] [Static] Property Get name [(arglist)] [As type]**

**[statements]**

**[name = expression]**

**[Exit Property]**

**[statements]**

**[name = expression]**

**End Property**

##### Declares the various parts of a Property Get procedure, which is used to obtain the value of a property. The optional Public, Private, and Friend keywords can be used to define the procedure's scope, and the optional Static keyword indicates that the procedure's local variables are preserved between calls to the procedure. The name argument specifies the name of the property to be retrieved and can be assigned a value (name = expression) that will be returned as the property's value. The data type of the property can be specified using the As type clause. The optional Exit Property can be used to exit the Property Get procedure immediately.

##### The optional list of arguments (arglist) defines the arguments that will be passed to the procedure. The arguments should use the following syntax:

##### **[Optional] [ByVal | ByRef] [ParamArray] varname[()] [As type] [= default value]**

##### where the Optional keyword can be used to specify that the argument is not required (default value assigns the argument's default value), ByVal and ByRef determine whether the argument should be passed by value or by reference (the default), and the ParamArray keyword specifies that the argument is an Optional array of Variant elements. ParamArray can only be used with the last argument in the argument list.

**Property Let Statement**

**[Public | Private | Friend] [Static] Property Let name ([arglist,] value)**

**[statements]**

**[Exit Property]**

**[statements]**

**End Property**

##### Declares the various parts of a Property Let procedure, which is used to assign a value to a property. The optional Public, Private, and Friend keywords can be used to define the procedure's scope, and the optional Static keyword indicates that the procedure's local variables are preserved between calls to the procedure. The name argument specifies the name of the property being referenced, and value indicates the value to be assigned to the property. The optional Exit Property can be used to exit the Property Let procedure immediately.

##### The optional list of arguments (arglist) defines the arguments that will be passed to the procedure. The arguments should use the following syntax:

##### **[Optional] [ByVal | ByRef] [ParamArray] varname[()] [As type] [= default value]**

##### where the Optional keyword can be used to specify that the argument is not required (default value assigns the argument's default value), ByVal and ByRef determine whether the argument should be passed by value or by reference (the default), and the ParamArray keyword specifies that the argument is an Optional array of Variant elements. ParamArray can only be used with the last argument in the argument list.

**Property Set Statement**

**[Public | Private | Friend] [Static] Property Set name ([arglist,] reference)**

**[statements]**

**[Exit Property]**

**[statements]**

**End Property**

##### Declares the various parts of a Property Set procedure, which is used to set a reference to an object. The optional Public, Private, and Friend keywords can be used to define the procedure's scope, and the optional Static keyword indicates that the procedure's local variables are preserved between calls to the procedure. The name argument specifies the name of the property being used, and reference indicates the object reference to be set to the property. The optional Exit Property can be used to exit the Property Set procedure immediately.

##### The optional list of arguments (arglist) defines the arguments that will be passed to the procedure. The arguments should use the following syntax:

**[Optional] [ByVal | ByRef] [ParamArray] varname[()] [As type] [= default value]**

##### where the Optional keyword can be used to specify that the argument is not required (default value assigns the argument's default value), ByVal and ByRef determine whether the argument should be passed by value or by reference (the default), and the ParamArray keyword specifies that the argument is an Optional array of Variant elements. ParamArray can only be used with the last argument in the argument list.

**Public Statement**

**Public [WithEvents] varname[([subscripts])] [As [New] type][,[WithEvents]**

**varname[([subscripts])] [As [New] type]]...**

##### Declares one or more public variables. The varname argument specifies the name of the variable being declared, and subscripts are the dimensions for an array variable. The optional As [New] type clause can be used to specify the variable's data type, with the New keyword enabling implicit creation of an object. The optional WithEvents keyword specifies that the variable being declared is an object variable used to respond to events triggered by an ActiveX object. The Public statement can only be used at module level, and variables declared with it can be used outside their own module.

**Put Statement**

**Put [#]filenumber, [recnumber], varname**

##### Writes data to the open disk file corresponding to the filenumber argument from a variable (varname). Put works with files open as Random or Binary, and a record number (recnumber) can be specified when writing data to a Random file. When using Binary files, recnumber can alternatively be used to specify the byte position at which the data is to be written.

**PV Function**

**PV(rate, nper, pmt[, fv[, type]])**

##### Returns a Double value that indicates the present value of an annuity based on a number (nper) of periodic fixed payments (pmt) and a fixed interest rate (rate). The optional fv argument specifies the future value or cash balance left after the final payment. The optional type argument specifies when payments are due (0 for end of the payment period, the default; 1 for beginning of the payment period).

**QBColor Function**

**QBColor(color)**

##### Returns a Long value that represents the RGB color code that corresponds to a given color number (0[nd]15) of the color palette used in Microsoft QuickBasic.

**RaiseEvent Function**

**RaiseEvent eventname [(argumentlist)]**

##### Triggers an event. The optional argumentlist specifies one or more comma-delimited arguments to be passed to the event procedure. The event procedure must be declared in the same module as the RaiseEvent function or an error will occur.

**\Randomize Statement**

**Randomize [number]**

##### Initializes the random number generator, using the optional number argument as a seed value.

**Rate Function**

**Rate(nper, pmt, pv[, fv[, type[, guess]]])**

##### Returns a Double value that indicates the fixed interest rate per period for an annuity based on a number (nper) of periodic fixed payments (pmt). The optional fv argument specifies the future value or cash balance left after the final payment. The optional type argument specifies when payments are due (0 for end of the payment period, the default; 1 for beginning of the payment period). The optional guess argument specifies an estimate value to be returned by Rate (default estimate is .1).

**ReDim Statement**

**ReDim [Preserve] varname(subscripts) [As type] [, varname(subscripts) [As type]]...**

##### Redimensions one or more dynamic array variables and reallocates their storage space. The optional Preserve keyword can be used to keep the contents of the array intact when it is being redimensioned. The varname argument is the name of the variable, and the optional As type clause indicates its data type. The subscripts are the dimensions of the array variable.

**Rem Statement**

**Rem comments**

##### Allows comments to be added to a program. Everything on the line after the Rem statement is ignored by Visual Basic. An apostrophe (`) can also be used in lieu of the Rem statement.

**Reset Statement**

**Reset**

##### Closes all files opened with the Open statement and writes any file buffer contents to disk.

**Resume Statement**

**Resume [0]**

**Resume Next**

**Resume line**

##### Resumes execution of a program when an error-handling routine is finished. Resume by itself causes execution to resume with the statement that caused the error or, if the error occurred in a called procedure, the statement that last called out of the error-handling procedure. Resume Next causes execution to resume with the statement immediately following the one that caused the error. Resume line transfers control to the line label or line number specified by the line argument.

**RGB Function**

**RGB(red, green, blue)**

##### Returns a Long value that represents an RGB color value as specified by the red, green, and blue color components passed to the RGB function. All color components should be Integers in the 0[nd]255 range.

**Right Function**

**Right(string, length)**

##### Returns a String value length characters long that is taken from the right side of a given string.

**RmDir Statement**

**RmDir path**

##### Removes the directory or folder specified by the path argument.

**Rnd Function**

**Rnd[(number)]**

##### Returns a Single value that contains a randomly generated number less than 1 but greater than or equal to zero. The optional number argument can be used to determine how Rnd generates the random number.

**RSet Statement**

**RSet stringvar = string**

##### Assigns a string value to a String variable (stringvar), right-aligning the string to the String variable.

**RTrim Function**

**RTrim(string)**

##### Returns a Variant of subtype String that contains a copy of a given string with any trailing spaces removed.

**SavePicture Statement**

**SavePicture picture, stringexpression**

##### Saves an graphic image from an object's Picture or Image property to a file. The picture argument specifies the control from which the graphics file is to be created (Picture or Image), and stringexpression specifies the path and filename to which the image is saved.

**SaveSetting Statement**

**SaveSetting appname, section, key, setting**

##### Saves or creates an application (appname) entry, section (section), key setting (key), and value (setting) in the System Registry.

**Second Function**

**Second(time)**

##### Returns a Variant of subtype Integer that represents the second (0[nd]59) of the time value specified by the time argument.

**Seek Function**

**Seek(filenumber)**

##### Returns a Long value that specifies the current record or byte position for the open file associated with filenumber. When dealing with Random files, Seek returns the number of the next record to be read or written. For all other file types, Seek returns a byte position.

**Seek Statement**

**Seek [#]filenumber, position**

##### Sets the record or byte position of the open file associated with file number.

**Select Case Statement**

**Select Case testexpression**

**[Case expressionlist-n**

**[statements-n]] ...**

**[Case Else**

**[elsestatements]]**

**End Select**

##### Evaluates an expression (testexpression) and, depending on the result, executes one or more statements (statements-n) that correspond to the expression's value (expressionlist-n). In other words, the value of testexpression is compared with one or more other values (expressionlist-n), and whichever matches gets its statements (statements-n) executed. If there are no matches, an optional Case Else set of statements (elsestatements) is executed.

**SendKeys Statement**

##### Generates one or more keystrokes as if they came from the keyboard. The string argument determines which keystrokes to send, and the optional Wait argument (Boolean) specifies whether keystrokes must be processed before control is returned to the procedure. False, the default value, means that control is returned to the procedure immediately after the keystrokes are sent.

**Set Statement**

**Set objectvar = {[New] objectexpression | Nothing}**

##### Assigns an object reference (objectexpression) to a variable or property (objectvar). The optional New keyword can be used to indicate that the object should be created implicitly. To disassociate objectvar with a specific object and free up the resources it is using, assign it the Nothing keyword.

**SetAttr Statement**

**SetAttr pathname, attributes**

##### Sets attributes for the file or directory specified by the pathname argument. The attributes argument can use several VB constants (vbNormal, vbReadOnly, vbHidden, vbSystem, vbDirectory, and vbArchive) that can be combined bitwise to determine which attributes are set.

**Sgn Function**

**Sgn(number)**

##### Returns a Variant of subtype Integer that represents the sign of a given number.

**Shell Function**

**Shell(pathname[, windowstyle])**

##### Runs the executable program specified by the pathname argument and returns a Variant of subtype Double that represents the program's task ID. If Shell is unsuccessful, it returns zero. The optional windowstyle argument determines the style of the window in which the shelled program runs.

**Sin Function**

**Sin(number)**

##### Returns a Double value that represents the sine of a given angle (as specified by the number argument).

**SLN Function**

**SLN(cost, salvage, life)**

##### Returns a Double value that represents the straight-line depreciation of an asset when given its initial cost, salvage value at the end of its useful life, and life span.

**Space Function**

**Space(number)**

##### Returns a Variant of subtype String that contains a number of spaces.

**Spc Function**

**Spc(n)**

##### Inserts a specified number of spaces (n) when writing or displaying text using the Print # statement or the Print method.

**Sqr Function**

**Sqr(number)**

##### Returns a Double value that represents the square root of a given number.

**Static Statement**

**Static varname[([subscripts])] [As [New] type] [, varname[([subscripts])]**

**[As [New] type] ...**

##### Declares one or more static variables, which retain their values as long as the program is running. The varname argument is the name of the variable, and the optional As [New] type indicates its data type. If the New keyword is used, then an implicit creation of the object is made. The optional subscripts are the dimensions of an array variable.

**Stop Statement**

**Stop**

##### Suspends program execution.

**Str Function**

**Str(number)**

##### Returns a Variant of subtype String that is a representation of a given number.

**StrComp Function**

**StrComp(string1, string2[, compare])**

##### Returns a Variant of subtype Integer that indicates the result of a comparison between two strings (string1 and string2). The optional compare argument specifies how strings are to be compared, with 0 for a binary comparison and 1 for a noncase- sensitive textual comparison.

**StrConv Function**

**StrConv(string, conversion)**

##### Returns a Variant of subtype String that has been converted from an original string as specified by the conversion argument. The conversion argument can use several VB constants to specify the type of conversion, such as vbUpperCase, vbLowerCase, and vbProperCase.

**String Function**

**String(number, character)**

##### Returns a Variant of subtype String that is of the length specified by number and is filled with a given character.

**Sub Statement**

**[Public | Private | Friend] [Static] Sub name [(arglist)]**

**[statements]**

**[Exit Sub]**

**[statements]**

**End Sub**

##### Declares the various parts of a Sub procedure. The optional Public, Private, and Friend keywords can be used to define the Sub's scope, and the optional Static keyword indicates that the procedure's local variables are preserved between calls to the Sub. The name argument specifies the name of the Sub procedure. The optional Exit Sub can be used to exit the Sub procedure immediately.

##### The optional list of arguments (arglist) defines the arguments that will be passed to the procedure. The arguments should use the following syntax:

##### **[Optional] [ByVal | ByRef] [ParamArray] varname[()] [As \_ type] [= default value]**

##### where the Optional keyword can be used to specify that the argument is not required (default value assigns the argument's default value), ByVal and ByRef determine whether the argument should be passed by value or by reference (the default), and the ParamArray keyword specifies that the argument is an Optional array of Variant elements. ParamArray can only be used with the last argument in the argument list.

**Switch Function**

**Switch(expr-1, value-1[, expr-2, value-2 ... [, expr-n, \_ value-n]])**

##### Evaluates a list of expressions (expr-1, expr-2...expr-n) and returns a Variant value that corresponds to the first expression that evaluates as True. If expr-1 is True, then Switch returns the value indicated by value-1; if expr-2 is True, then Switch returns the value indicated by value-2; and so on.

**SYD Function**

**SYD(cost, salvage, life, period)**

##### Returns a Double value that represents the sum-of-years' digits depreciation of an asset when given its initial cost, salvage value at the end of its useful life, life span, and period for which depreciation is calculated.

**Tab Function**

**Tab(n)**

##### Positions output to a given column (n) when writing or displaying text using the Print # statement or the Print method.

**Tan Function**

**Tan(number)**

##### Returns a Double value that represents the tangent of a given angle (specified by the number argument).

**Time Function**

**Time**

##### Returns a Variant of subtype Date that contains the current system time.

**Time Statement**

**Time = time**

##### Sets the system time to the time specified by the time argument.

**Timer Function**

**Timer**

##### Returns a Single value that represents the number of seconds that have elapsed since midnight.

**TimeSerial Function**

**TimeSerial(hour, minute, second)**

##### Returns a Variant of subtype Date that represents a time as specified by the hour, minute, and second arguments.

**TimeValue Function**

**TimeValue(time)**

##### Returns a Variant of subtype Date that is derived from the time value specified by the time argument.

**Trim Function**

**Trim(string)**

##### Returns a Variant of subtype String that contains a copy of a given string with any leading and trailing spaces removed.

**Type Statement**

**[Private | Public] Type varname**

**elementname [([subscripts])] As type**

**[elementname [([subscripts])] As type]**

**...**

**End Type**

##### Defines a user-defined type (UDT) structure that contains one or more elements (elementname). The optional Public and Private keywords specify the UDT's scope, and varname specifies the UDT's name. Elements can be arrays (by specifying subscripts), and their data type must be defined using the As type clause. The Type statement can only be used at module level.

**TypeName Function**

**TypeName(varname)**

##### Returns a String value that indicates the data type of a given variable (varname). Possible return values are: Byte, Integer, Long, Single, Double, Currency, Decimal, Date, String, Boolean, Error, Empty, Null, Object, Unknown, Nothing, or an object type.

**UBound Function**

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

##### Returns a Long value that represents the largest subscript for a dimensioned array (arrayname). For multidimensional arrays, the dimension argument can be included to specify which dimension should be used.

**UCase Function**

**UCase(string)**

##### Converts a string to all uppercase characters and returns a new String value.

**Unload Statement**

**Unload object**

##### Unloads an object (such as a form or control) from memory and frees up any resources being used by the object.

**Unlock Statement**

**Unlock [#]filenumber[, recordrange]**

##### Removes locking that prevents another process from accessing all or part of the open file that corresponds to the filenumber argument. The recordrange argument refers to a range of records (or bytes) that are to be unlocked and should use the syntax:

**recnumber | [start] To end**

##### where recnumber is the record number (for Random files) or byte position (for Binary files) where unlocking should begin. Alternatively, the starting and ending record numbers or bytes to be unlocked can be specified using the start and end arguments.

**Val Function**

**Val(string)**

##### Returns the numeric value of a string. The data type that is returned by Val depends on the kind of numeric value the string contains. If the string does not contain a numeric value, then Val returns zero.

**VarType Function**

**VarType(varname)**

##### Returns an Integer value that represents the subtype of the variable specified by varname. Several VB constants are used to define the data type values returned by the VarType function, including: vbEmpty, vbNull, vbInteger, vbLong, vbSingle, vbDouble, vbCurrency, vbDate, vbString, vbObject, vbError, vbBoolean, vbVariant, vbDataObject, vbDecimal, vbByte, and vbArray.

**Weekday Function**

**Weekday(date, [firstdayofweek])**

##### Returns a Variant of subtype Integer that represents the day of the week for a given date. Weekday returns a 1 for Sunday, 2 for Monday, and so on. The optional firstdayofweek argument can be used to specify the first day of the week. If firstdayofweek is not specified, then Sunday (1) is assumed.

**While...Wend Statement**

**While condition**

**[statements]**

**Wend**

##### Repeats one or more statements while a condition remains True. When the condition becomes False, then control is passed to the line of code immediately following the While...Wend structure.

**Width # Statement**

**Width #filenumber, width**

##### Assigns an output line width (in characters) for the open file associated with filenumber.

**With Statement**

**With object**

**[statements]**

**End With**

##### Executes one or more statements on a single object or user-defined type.

**Write # Statement**

**Write #filenumber, [outputlist]**

##### Writes data to the open sequential file associated with the filenumber argument. The varlist argument should contain one or more comma-delimited variables that contain the data to be written to the file.

**Year Function**

**Year(date)**

##### Returns a Variant that represents the year for the date value specified by the date argument.

# CREATED BY :MTM SUHAIL

# MUTHIRATHODIKA

# PARAKKAPADAM

# POOKKOOTTUMPADAM(PO)

# PIN:679332

# PH:9047717702