Visual Basic for Applications Reference

Visual Studio 6.0

Hex Function

See Also Example Specifics

Returns a String representing the hexadecimal value of a number.

Syntax

Hex(*number*)

The required *number* argument is any valid numeric expression or string 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.

Visual Basic for Applications Reference

Hex Function Example

This example uses the **Hex** function to return the hexadecimal value of a number.

```
Dim MyHex
MyHex = Hex(5)   ' Returns 5.
MyHex = Hex(10)   ' Returns A.
MyHex = Hex(459)   ' Returns 1CB.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Hour Function

See Also Example Specifics

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 23, inclusive, representing the hour of the day.

Syntax

Hour(*time***)**

The required *time* argument is any Variant, numeric expression, string expression, or any combination, that can represent a time. If *time* contains Null, **Null** is returned.

Visual Basic for Applications Reference

Hour Function Example

This example uses the **Hour** function to obtain the hour from a specified time. In the development environment, the time literal is displayed in short time format using the locale settings of your code.

```
Dim MyTime, MyHour
MyTime = #4:35:17 PM#   ' Assign a time.
MyHour = Hour(MyTime)   ' MyHour contains 16.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IIf Function

See Also Example Specifics

Returns one of two parts, depending on the evaluation of an expression.

Syntax

IIf(*expr*, *truepart*, *falsepart*)

The **IIf** function syntax has these named arguments:

Part	Description
expr	Required. Expression you want to evaluate.
truepart	Required. Value or expression returned if ***expr*** is **True**.
falsepart	Required. Value or expression returned if ***expr*** is **False**.

Remarks

IIf always evaluates both ***truepart*** and ***falsepart***, even though it returns only one of them. Because of this, you should watch for undesirable side effects. For example, if evaluating ***falsepart*** results in a division by zero error, an error occurs even if ***expr*** is **True**.

Visual Basic for Applications Reference

IIf Function Example

This example uses the **IIf** function to evaluate the `TestMe` parameter of the `CheckIt` procedure and returns the word "Large" if the amount is greater than 1000; otherwise, it returns the word "Small".

```
Function CheckIt (TestMe As Integer)
   CheckIt = IIf(TestMe > 1000, "Large", "Small")
End Function
```

Visual Basic for Applications Reference

Visual Studio 6.0

IMEStatus Function

See Also Example Specifics

Returns an Integer specifying the current Input Method Editor (IME) mode of Microsoft Windows; available in East Asian versions only.

Syntax

IMEStatus

Return Values

The return values for the Japanese locale are as follows:

Constant	Value	Description
vbIMEModeNoControl	0	Don't control IME (default)
vbIMEModeOn	1	IME on
vbIMEModeOff	2	IME off
vbIMEModeDisable	3	IME disabled
vbIMEModeHiragana	4	Full-width Hiragana mode
vbIMEModeKatakana	5	Full-width Katakana mode
vbIMEModeKatakanaHalf	6	Half-width Katakana mode
vbIMEModeAlphaFull	7	Full-width Alphanumeric mode
vbIMEModeAlpha	8	Half-width Alphanumeric mode

The return values for the Korean locale are as follows:

Constant	Value	Description
vbIMEModeNoControl	0	Don't control IME(default)

vbIMEModeAlphaFull	7	Full-width Alphanumeric mode
vbIMEModeAlpha	8	Half-width Alphanumeric mode
vbIMEModeHangulFull	9	Full-width Hangul mode
vbIMEModeHangul	10	Half-width Hangul mode

The return values for the Chinese locale are as follows:

Constant	Value	Description
vbIMEModeNoControl	0	Don't control IME (default)
vbIMEModeOn	1	IME on
vbIMEModeOff	2	IME off

Visual Basic for Applications Reference

Visual Studio 6.0

Input Function

See Also Example Specifics

Returns String containing characters from a file opened in **Input** or **Binary** mode.

Syntax

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

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

Part	Description
number	Required. Any valid numeric expression specifying the number of characters to return.
filenumber	Required. Any valid file number.

Remarks

Data read with the **Input** function is usually written to a file with **Print #** or **Put**. Use this function only with files opened in **Input** or **Binary** mode.

Unlike the **Input #** statement, the **Input** function returns all of the characters it reads, including commas, carriage returns, linefeeds, quotation marks, and leading spaces.

With files opened for **Binary** access, an attempt to read through the file using the **Input** function until **EOF** returns **True** generates an error. Use the **LOF** and **Loc** functions instead of **EOF** when reading binary files with **Input**, or use **Get** when using the **EOF** function.

> **Security Note** When reading from files, do not make decisions about the contents of the file based on the file name extension. For example, a file named Form1.vb may not be a Visual Basic source file.

Note Use the **InputB** function for byte data contained within text files. With **InputB**, *number* specifies the number of bytes to return rather than the number of characters to return.

Visual Basic for Applications Reference

Input Function Example

This example uses the **Input** function to read one character at a time from a file and print it to the **Immediate** window. This example assumes that `TESTFILE` is a text file with a few lines of sample data.

```
Dim MyChar
Open "TESTFILE" For Input As #1   ' Open file.
Do While Not EOF(1)   ' Loop until end of file.
   MyChar = Input(1, #1)   ' Get one character.
   Debug.Print MyChar   ' Print to the Immediate window.
Loop
Close #1   ' Close file.
```

Visual Basic for Applications Reference

Visual Studio 6.0

InputBox Function

See Also Example Specifics

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

Syntax

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

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

Part	Description
prompt	Required. 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 returnlinefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.
title	Optional. 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	Optional. 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	Optional. 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	Optional. 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	Optional. 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	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

Remarks

When both ***helpfile*** and ***context*** are provided, the user can press F1 to view the Help topic corresponding to the ***context***. Some host applications, for example, Microsoft Excel, also automatically add a **Help** button 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 ("").

Note To specify more than the first named argument, you must use **InputBox** in an expression. To omit some positional arguments, you must include the corresponding comma delimiter.

Visual Basic for Applications Reference

InputBox Function Example

This example shows various ways to use the **InputBox** function to prompt the user to enter a value. If the x and y positions are omitted, the dialog box is automatically centered for the respective axes. The variable `MyValue` contains the value entered by the user if the user clicks **OK** or presses the ENTER key . If the user clicks **Cancel**, a zero-length string is returned.

```
Dim Message, Title, Default, MyValue
Message = "Enter a value between 1 and 3"   ' Set prompt.
Title = "InputBox Demo"   ' Set title.
Default = "1"   ' Set default.
' Display message, title, and default value.
MyValue = InputBox(Message, Title, Default)

' Use Helpfile and context. The Help button is added automatically.
MyValue = InputBox(Message, Title, , , , "DEMO.HLP", 10)

' Display dialog box at position 100, 100.
MyValue = InputBox(Message, Title, Default, 100, 100)
```

Visual Basic for Applications Reference

Visual Studio 6.0

InStr Function

See Also Example Specifics

Returns a **Variant** (**Long**) specifying 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 sought.
compare	Optional. Specifies the type of string comparison. If ***compare*** is Null, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison. Specify a valid LCID (LocaleID) to use locale-specific rules in the comparison.

Settings

The *compare* argument settings are:

Constant	Value	Description
vbUseCompareOption	-1	Performs a comparison using the setting of the **Option Compare** statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Return 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 > ***string2***	0

Remarks

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.

Visual Basic for Applications Reference

InStr Function Example

This example uses the **InStr** function to return the position of the first occurrence of one string within another.

```
Dim SearchString, SearchChar, MyPos
SearchString ="XXpXXpXXPXXP"   ' String to search in.
SearchChar = "P"   ' Search for "P".

' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)

' A binary comparison starting at position 1. Returns 9.
MyPos = Instr(1, SearchString, SearchChar, 0)

' Comparison is binary by default (last argument is omitted).
MyPos = Instr(SearchString, SearchChar)   ' Returns 9.

MyPos = Instr(1, SearchString, "W")   ' Returns 0.
```

Visual Basic for Applications Reference

Visual Studio 6.0

InStrRev Function

See Also Example Specifics

Description

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

Syntax

InstrRev(*stringcheck*, *stringmatch*[, *start*[, *compare*]])

The **InstrRev** function syntax has these named arguments:

Part	Description
stringcheck	Required. String expression being searched.
stringmatch	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
vbUseCompareOption	1	Performs a comparison using the setting of the **Option Compare** statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Return Values

InStrRev returns the following values:

If	InStrRev returns
stringcheck is zero-length	0
stringcheck is **Null**	**Null**
stringmatch is zero-length	*start*
stringmatch is **Null**	**Null**
stringmatch is not found	0
stringmatch is found within ***stringcheck***	Position at which match is found
start > **Len(*stringmatch*)**	0

Remarks

Note that the syntax for the **InstrRev** function is not the same as the syntax for the **Instr** function.

Visual Basic for Applications Reference

Visual Studio 6.0

Int, Fix Functions

See Also Example Specifics

Returns the integer portion of a number.

Syntax

Int(*number***)**

Fix(*number***)**

The required *number* argument is a Double or 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))
```

Visual Basic for Applications Reference

Int Function, Fix Function Example

This example illustrates how the **Int** and **Fix** functions return integer portions of numbers. In the case of a negative number argument, the **Int** function returns the first negative integer less than or equal to the number; the **Fix** function returns the first negative integer greater than or equal to the number.

```
Dim MyNumber
MyNumber = Int(99.8)   ' Returns 99.
MyNumber = Fix(99.2)   ' Returns 99.

MyNumber = Int(-99.8)   ' Returns -100.
MyNumber = Fix(-99.8)   ' Returns -99.

MyNumber = Int(-99.2)   ' Returns -100.
MyNumber = Fix(-99.2)   ' Returns -99.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IPmt Function

See Also Example Specifics

Returns a Double specifying the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax

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

The **IPmt** function has these named arguments:

Part	Description
rate	Required. **Double** specifying interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
per	Required. **Double** specifying payment period in the range 1 through ***nper***.
nper	Required. **Double** specifying total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
pv	Required. **Double** specifying present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
fv	Optional. Variant specifying future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type	Optional. **Variant** specifying when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

The ***rate*** and ***nper*** arguments must be calculated using payment periods expressed in the same units. For example, if ***rate*** is calculated using months, ***nper*** must also be calculated using months.

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

Visual Basic for Applications Reference

IPmt Function Example

This example uses the **IPmt** function to calculate how much of a payment is interest when all the payments are of equal value. Given are the interest percentage rate per period (`APR / 12`), the payment period for which the interest portion is desired (`Period`), the total number of payments (`TotPmts`), the present value or principal of the loan (`PVal`), the future value of the loan (`FVal`), and a number that indicates whether the payment is due at the beginning or end of the payment period (`PayType`).

```
Dim FVal, Fmt, PVal, APR, TotPmts, PayType, Period, IntPmt, TotInt, Msg
Const ENDPERIOD = 0, BEGINPERIOD = 1   ' When payments are made.
FVal = 0   ' Usually 0 for a loan.
Fmt = "###,###,##0.00"   ' Define money format.
PVal = InputBox("How much do you want to borrow?")
APR = InputBox("What is the annual percentage rate of your loan?")
If APR > 1 Then APR = APR / 100   ' Ensure proper form.
TotPmts = InputBox("How many monthly payments?")
PayType = MsgBox("Do you make payments at end of the month?", vbYesNo)
If PayType = vbNo Then PayType = BEGINPERIOD Else PayType = ENDPERIOD
For Period = 1 To TotPmts   ' Total all interest.
   IntPmt = IPmt(APR / 12, Period, TotPmts, -PVal, FVal, PayType)
   TotInt = TotInt + IntPmt
Next Period
Msg = "You'll pay a total of " & Format(TotInt, Fmt)
Msg = Msg & " in interest for this loan."
MsgBox Msg   ' Display results.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IRR Function

See Also Example Specifics

Returns a Double specifying the internal rate of return for a series of periodic cash flows (payments and receipts).

Syntax

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

The **IRR** function has these named arguments:

Part	Description
***values*()**	Required. Array of **Double** specifying cash flow values. The array must contain at least one negative value (a payment) and one positive value (a receipt).
guess	Optional. Variant specifying value you estimate will be returned by **IRR**. If omitted, ***guess*** is 0.1 (10 percent).

Remarks

The internal rate of return is the interest rate received for an investment consisting of payments and receipts that occur at regular intervals.

The **IRR** function uses the order of values within the array to interpret the order of payments and receipts. Be sure to enter your payment and receipt values in the correct sequence. The cash flow for each period doesn't have to be fixed, as it is for an annuity.

IRR is calculated by iteration. Starting with the value of ***guess***, **IRR** cycles through the calculation until the result is accurate to within 0.00001 percent. If **IRR** can't find a result after 20 tries, it fails.

Visual Basic for Applications Reference

IRR Function Example

In this example, the **IRR** function returns the internal rate of return for a series of 5 cash flows contained in the array `Values()`. The first array element is a negative cash flow representing business start-up costs. The remaining 4 cash flows represent positive cash flows for the subsequent 4 years. `Guess` is the estimated internal rate of return.

```
Dim Guess, Fmt, RetRate, Msg
Static Values(5) As Double   ' Set up array.
Guess = .1   ' Guess starts at 10 percent.
Fmt = "#0.00"   ' Define percentage format.
Values(0) = -70000   ' Business start-up costs.
' Positive cash flows reflecting income for four successive years.
Values(1) = 22000 : Values(2) = 25000
Values(3) = 28000 : Values(4) = 31000
RetRate = IRR(Values(), Guess) * 100   ' Calculate internal rate.
Msg = "The internal rate of return for these five cash flows is "
Msg = Msg & Format(RetRate, Fmt) & " percent."
MsgBox Msg   ' Display internal return rate.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsArray Function

See Also Example Specifics

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

Syntax

IsArray(*varname***)**

The required *varname* argument is an identifier specifying a variable.

Remarks

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

Visual Basic for Applications Reference

IsArray Function Example

This example uses the **IsArray** function to check if a variable is an array.

```
Dim MyArray(1 To 5) As Integer, YourArray, MyCheck   ' Declare array variables.
YourArray = Array(1, 2, 3)   ' Use Array function.
MyCheck = IsArray(MyArray)   ' Returns True.
MyCheck = IsArray(YourArray)   ' Returns True.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsDate Function

See Also Example Specifics

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

Syntax

IsDate(*expression***)**

The required *expression* argument is a Variant containing a date expression or string expression recognizable as a date or time.

Remarks

IsDate returns **True** if the expression is a date or is recognizable as 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.

Visual Basic for Applications Reference

IsDate Function Example

This example uses the **IsDate** function to determine if an expression can be converted to a date.

```
Dim MyDate, YourDate, NoDate, MyCheck
MyDate = "February 12, 1969": YourDate = #2/12/69#: NoDate = "Hello"
MyCheck = IsDate(MyDate)   ' Returns True.
MyCheck = IsDate(YourDate)   ' Returns True.
MyCheck = IsDate(NoDate)   ' Returns False.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsEmpty Function

See Also Example Specifics

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

Syntax

IsEmpty(*expression***)**

The required *expression* argument is a Variant containing a numeric or string 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. **IsEmpty** only returns meaningful information for variants.

IsEmpty Function Example

This content is no longer actively maintained. It is provided as is, for anyone who may still be using these technologies, with no warranties or claims of accuracy with regard to the most recent product version or service release.

This example sorts the data in the first column on Sheet1 and then deletes any rows that contain duplicate data.

```
Worksheets("Sheet1").Range("A1").Sort _
    key1:=Worksheets("Sheet1").Range("A1")
Set currentCell = Worksheets("Sheet1").Range("A1")
Do While Not IsEmpty(currentCell)
    Set nextCell = currentCell.Offset(1, 0)
    If nextCell.Value = currentCell.Value Then
        currentCell.EntireRow.Delete
    End If
    Set currentCell = nextCell
Loop
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsError Function

See Also Example Specifics

Returns a **Boolean** value indicating whether an expression is an error value.

Syntax

IsError(*expression***)**

The required *expression* argument can be any valid expression.

Remarks

Error values are created by converting real numbers to error values using the **CVErr** function. The **IsError** function is used to determine if a numeric expression represents an error. **IsError** returns **True** if the *expression* argument indicates an error; otherwise, it returns **False**.

Visual Basic for Applications Reference

IsError Function Example

This example uses the **IsError** function to check if a numeric expression is an error value. The **CVErr** function is used to return an **Error Variant** from a user-defined function. Assume `UserFunction` is a user-defined function procedure that returns an error value; for example, a return value assigned with the statement `UserFunction = CVErr(32767)`, where 32767 is a user-defined number.

```
Dim ReturnVal, MyCheck
ReturnVal = UserFunction()
MyCheck = IsError(ReturnVal)   ' Returns True.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsMissing Function

See Also Example Specifics

Returns a **Boolean** value indicating whether an optional **Variant** argument has been passed to a procedure.

Syntax

IsMissing(*argname***)**

The required *argname* argument contains the name of an optional **Variant** procedure argument.

Remarks

Use the **IsMissing** function to detect whether or not optional **Variant** arguments have been provided in calling a procedure. **IsMissing** returns **True** if no value has been passed for the specified argument; otherwise, it returns **False**. If **IsMissing** returns **True** for an argument, use of the missing argument in other code may cause a user-defined error. If **IsMissing** is used on a **ParamArray** argument, it always returns **False**. To detect an empty **ParamArray**, test to see if the arrays upper bound is less than its lower bound.

Note **IsMissing** does not work on simple data types (such as **Integer** or **Double**) because, unlike **Variants**, they don't have a provision for a "missing" flag bit. Because of this, the syntax for typed optional arguments allows you to specify a default value. If the argument is omitted when the procedure is called, then the argument will have this default value, as in the example below:

```
Sub MySub(Optional MyVar As String = "specialvalue")
    If MyVar = "specialvalue" Then
       ' MyVar was omitted.
    Else
    ...
End Sub
```

In many cases you can omit the `If MyVar` test entirely by making the default value equal to the value you want `MyVar` to contain if the user omits it from the function call. This makes your code more concise and efficient.

Visual Basic for Applications Reference

IsMissing Function Example

This example uses the **IsMissing** function to check if an optional argument has been passed to a user-defined procedure. Note that **Optional** arguments can now have default values and types other than **Variant**.

```
Dim ReturnValue
' The following statements call the user-defined function procedure.
ReturnValue = ReturnTwice()   ' Returns Null.
ReturnValue = ReturnTwice(2)   ' Returns 4.

' Function procedure definition.
Function ReturnTwice(Optional A)
   If IsMissing(A) Then
      ' If argument is missing, return a Null.
      ReturnTwice = Null
   Else
      ' If argument is present, return twice the value.
      ReturnTwice = A * 2
   End If
End Function
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsNull Function

See Also Example Specifics

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

Syntax

IsNull(*expression***)**

The required *expression* argument is a Variant containing a numeric expression or string expression.

Remarks

IsNull returns **True** if *expression* is **Null**; 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 **Variant** 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**.

IsNull Function Example

This content is no longer actively maintained. It is provided as is, for anyone who may still be using these technologies, with no warranties or claims of accuracy with regard to the most recent product version or service release.

This example creates a list of registered functions, placing each one in a separate row on Sheet1. Column A contains the full path and file name of the DLL or code resource, column B contains the function name, and column C contains the code for the argument data type.

```
theArray = Application.RegisteredFunctions
If IsNull(theArray) Then
    MsgBox "No registered functions"
Else
    For i = LBound(theArray) To UBound(theArray)
        For j = 1 To 3
            Worksheets("Sheet1").Cells(i, j).Formula = _
                theArray(i, j)
        Next j
    Next i
End If
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsNumeric Function

See Also Example Specifics

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

Syntax

IsNumeric(*expression***)**

The required *expression* argument is a Variant containing a numeric expression or string 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.

Visual Basic for Applications Reference

IsNumeric Function Example

This example uses the **IsNumeric** function to determine if a variable can be evaluated as a number.

```
Dim MyVar, MyCheck
MyVar = "53"   ' Assign value.
MyCheck = IsNumeric(MyVar)   ' Returns True.

MyVar = "459.95"   ' Assign value.
MyCheck = IsNumeric(MyVar)   ' Returns True.

MyVar = "45 Help"   ' Assign value.
MyCheck = IsNumeric(MyVar)   ' Returns False.
```

Visual Basic for Applications Reference

Visual Studio 6.0

IsObject Function

See Also Example Specifics

Returns a **Boolean** value indicating whether an identifier represents an object variable.

Syntax

IsObject(*identifier***)**

The required *identifier* argument is a variable name.

Remarks

IsObject is useful only in determining whether a Variant is of **VarType vbObject**. This could occur if the **Variant** actually references (or once referenced) an object, or if it contains **Nothing.**

IsObject returns **True** if *identifier* is a variable declared with Object type or any valid class type, or if *identifier* is a **Variant** of **VarType vbObject**, or a user-defined object; otherwise, it returns **False**. **IsObject** returns **True** even if the variable has been set to **Nothing**.

Use error trapping to be sure that an object reference is valid.

Visual Basic for Applications Reference

IsObject Function Example

This example uses the **IsObject** function to determine if an identifier represents an object variable. `MyObject` and `YourObject` are object variables of the same type. They are generic names used for illustration purposes only.

```
Dim MyInt As Integer, YourObject, MyCheck   ' Declare variables.
Dim MyObject As Object
Set YourObject = MyObject   ' Assign an object reference.
MyCheck = IsObject(YourObject)   ' Returns True.
MyCheck = IsObject(MyInt)   ' Returns False.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Join Function

See Also Example Specifics

Description

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

Syntax

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

The **Join** function syntax has these named arguments:

Part	Description
sourcearray	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.

Visual Basic for Applications Reference

Visual Studio 6.0

LBound Function

See Also Example Specifics

Returns a Long containing 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	Required. Name of the array variable; follows standard variable naming conventions.
dimension	Optional; **Variant** (**Long**). 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.

LBound returns the values in the following table for an array with the following dimensions:

```
Dim A(1 To 100, 0 To 3, -3 To 4)
```

Statement	Return Value
`LBound(A, 1)`	1
`LBound(A, 2)`	0
`LBound(A, 3)`	-3

The default lower bound for any dimension is either 0 or 1, depending on the setting of the **Option Base** statement. The base of an array created with the **Array** function is zero; it is unaffected by **Option Base**.

Arrays for which dimensions are set using the **To** clause in a **Dim**, **Private**, **Public**, **ReDim**, or **Static** statement can have any integer value as a lower bound.

LBound Function Example

This example writes the elements of the first custom list in column one on Sheet1.

```
listArray = Application.GetCustomListContents(1)
For i = LBound(listArray, 1) To UBound(listArray, 1)
    Worksheets("sheet1").Cells(i, 1).Value = listArray(i)
Next i
```

This example assumes that you used an external data source to create a PivotTable report on Sheet1 in the active workbook. The example inserts the SQL connection string and query string into a new worksheet.

```
Set newSheet = ActiveWorkbook.Worksheets.Add
sdArray = Worksheets("Sheet1").UsedRange.PivotTable.SourceData
For i = LBound(sdArray) To UBound(sdArray)
    newSheet.Cells(i, 1) = sdArray(i)
Next i
```

Visual Basic for Applications Reference

Visual Studio 6.0

LCase Function

See Also Example Specifics

Returns a String that has been converted to lowercase.

Syntax

LCase(*string*)

The required *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.

Visual Basic for Applications Reference

LCase Function Example

This example uses the **LCase** function to return a lowercase version of a string.

```
Dim UpperCase, LowerCase
Uppercase = "Hello World 1234"   ' String to convert.
Lowercase = Lcase(UpperCase)   ' Returns "hello world 1234".
```

Visual Basic for Applications Reference

Visual Studio 6.0

Left Function

See Also Example Specifics

Returns a **Variant** (**String**) containing a specified number of characters from the left side of a string.

Syntax

Left(***string***, ***length***)

The **Left** function syntax has these named arguments:

Part	Description
string	Required. String expression from which the leftmost characters are returned. If ***string*** contains Null, Null is returned.
length	Required; **Variant** (**Long**). 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 Use the **LeftB** function with byte data contained in a string. Instead of specifying the number of characters to return, ***length*** specifies the number of bytes.

Visual Basic for Applications Reference

Left Function Example

This example uses the **Left** function to return a specified number of characters from the left side of a string.

```
Dim AnyString, MyStr
AnyString = "Hello World"   ' Define string.
MyStr = Left(AnyString, 1)   ' Returns "H".
MyStr = Left(AnyString, 7)   ' Returns "Hello W".
MyStr = Left(AnyString, 20)   ' Returns "Hello World".
```

Visual Basic for Applications Reference

Visual Studio 6.0

Len Function

See Also Example Specifics

Returns a Long containing 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. If *varname* is a Variant, **Len** treats it the same as a **String** and always returns the number of characters it contains.

Remarks

One (and only one) of the two possible arguments must be specified. With user-defined types, **Len** returns the size as it will be written to the file.

Note Use the **LenB** function with byte data contained in a string, as in double-byte character set (DBCS) languages. Instead of returning the number of characters in a string, **LenB** returns the number of bytes used to represent that string. With user-defined types, **LenB** returns the in-memory size, including any padding between elements. For sample code that uses **LenB**, see the second example in the example topic.

Note **Len** may not be able to determine the actual number of storage bytes required when used with variable-length strings in user-defined data types.

Visual Basic for Applications Reference

Len Function Example

The first example uses **Len** to return the number of characters in a string or the number of bytes required to store a variable. The **Type...End Type** block defining `CustomerRecord` must be preceded by the keyword **Private** if it appears in a class module. In a standard module, a **Type** statement can be **Public**.

```
Type CustomerRecord   ' Define user-defined type.
   ID As Integer   ' Place this definition in a
   Name As String * 10   ' standard module.
   Address As String * 30
End Type

Dim Customer As CustomerRecord   ' Declare variables.
Dim MyInt As Integer, MyCur As Currency
Dim MyString, MyLen
MyString = "Hello World"   ' Initialize variable.
MyLen = Len(MyInt)   ' Returns 2.
MyLen = Len(Customer)   ' Returns 42.
MyLen = Len(MyString)   ' Returns 11.
MyLen = Len(MyCur)   ' Returns 8.
```

The second example uses **LenB** and a user-defined function (**LenMbcs**) to return the number of byte characters in a string if ANSI is used to represent the string.

```
Function LenMbcs (ByVal str as String)
   LenMbcs = LenB(StrConv(str, vbFromUnicode))
End Function

Dim MyString, MyLen
MyString = "ABc"
' Where "A" and "B" are DBCS and "c" is SBCS.
MyLen = Len(MyString)
' Returns 3 - 3 characters in the string.
MyLen = LenB(MyString)
' Returns 6 - 6 bytes used for Unicode.
MyLen = LenMbcs(MyString)
' Returns 5 - 5 bytes used for ANSI.
```

Visual Basic Reference

Visual Studio 6.0

LoadPicture Function

See Also Example

Loads a graphic into a forms **Picture** property, a **PictureBox** control, or an **Image** control.

Syntax

LoadPicture([*filename*], [*size*], [*colordepth*],[*x*,*y*]**)**

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

Part	Description
filename	Optional. String expression specifying a filename. Can include folder and drive. If no filename is specified LoadPicture clears the Image or PictureBox control.
size	Optional variant. If *filename* is a cursor or icon file, specifies the desired image size.
colordepth	Optional variant. If *filename* is a cursor or icon file, specifies the desired color depth.
x	Optional variant, required if *y* is used. If *filename* is a cursor or icon file, specifies the width desired. In a file containing multiple separate images, the best possible match is used if an image of that size is not available. X and y values are only used when *colordepth* is set to **vbLPCustom**. For icon files 255 is the maximum possible value.
y	Optional variant, required if *x* is used. If *filename* is a cursor or icon file, specifies the height desired. In a file containing multiple separate images, the best possible match is used if an image of that size is not available. For icon files 255 is the maximum possible value.

Settings

The settings for *size* are:

Constant	Value	Description
vbLPSmall	0	System small icon.

vbLPLarge	1	System large icon size, as determined by the video driver.
vbLPSmallShell	2	Shell small icon size, as determined by the Caption Buttons size setting on the **Appearance** tab in the Control Panel **Display Properties** dialog box.
vbLPLargeShell	3	Shell large icon size, as determined by the Icon size setting on the **Appearance** tab in the Control Panel **Display Properties** dialog box.
vbLPCustom	4	Custom size, values provided by *x* and *y* arguments

The settings for *colordepth* are:

Constant	Value	Description
vbLPDefault	0	Best available match if the specified file is used.
vbLPMonochrome	1	2 colors.
vbLPVGAColor	2	16 colors.
vbLPColor	3	256 colors.

Remarks

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

Graphics are cleared from forms, picture boxes, and image controls by assigning **LoadPicture** with no argument.

To load graphics for display in a **PictureBox** control, **Image** control, or as the background of a form, the return value of **LoadPicture** must be assigned to the **Picture** property of the object on which the picture is displayed. For example:

```
Set Picture = LoadPicture("PARTY.BMP")
Set Picture1.Picture = LoadPicture("PARTY.BMP")
```

To assign an icon to a form, set the return value of the **LoadPicture** function to the **Icon** property of the **Form** object:

```
Set Form1.Icon = LoadPicture("MYICON.ICO")
```

Icons can also be assigned to the **DragIcon** property of all controls except **Timer** controls and **Menu** controls. For example:

```
Set Command1.DragIcon = LoadPicture("MYICON.ICO")
```

Load a graphics file into the system Clipboard using **LoadPicture** as follows:

```
Clipboard.SetData LoadPicture("PARTY.BMP")
```

Visual Basic Reference

LoadPicture Function Example

This example uses the **LoadPicture** function to load a picture into a **PictureBox** control and to clear the picture from the control. To try this example, add a **PictureBox** control to a **Form** object, paste the code into the Declarations section of the **Form**, and then run the example and click the **Form**.

```
Private Sub Form_Click ()
   Dim Msg as String   ' Declare variables.
   On Error Resume Next   ' Set up error handling.
   Height = 3990
   Width = 4890   ' Set height and width.
   Picture1.Picture = LoadPicture("PAPER.CUR", vbLPCustom, vbLPColor, 32, 32)   ' Load cursor.
   If Err Then
      Msg = "Couldn't find the .cur file."
      MsgBox Msg   ' Display error message.
      Exit Sub   ' Quit if error occurs.
   End If
   Msg = "Choose OK to clear the bitmap from the picturebox."
   MsgBox Msg
   Picture1.Picture = LoadPicture()   'Clear the picturebox.
End Sub
```

Visual Basic Reference

Visual Studio 6.0

LoadResData Function

See Also Example Applies To

Loads data of several possible types from a resource (.res) file and returns a **Byte** array.

Syntax

LoadResData(*index*, *format***)**

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

Part	Description
index	Required. Integer or string specifying the identifier (ID) of the data in the resource file. The resource whose ID is 1 is reserved for the application icon.
format	Required. Value that specifies the original format of the data being returned, as described in Settings. Value can also be the string name of a user-defined resource.

Settings

The settings for *format* are:

Setting	Description
1	Cursor resource
2	Bitmap resource
3	Icon resource
4	Menu resource
5	Dialog box
6	String resource
7	Font directory resource
8	Font resource

9	Accelerator table
10	User-defined resource
12	Group cursor
14	Group icon

Remarks

The data that **LoadResData** loads from the resource file can be up to 64K.

Using **LoadResData** with a bitmap, icon, or cursor resource type returns a string containing the actual bits in the resource. If you want to use the actual bitmap, icon, or resource, use the **LoadResPicture** function.

Using **LoadResData** is useful for localizing a Visual Basic application because the resources that need to be translated are isolated in one resource file and there is no need to access the source code or recompile the application.

Visual Basic Reference

Visual Studio 6.0

LoadResPicture Function

See Also Example Applies To

Loads a bitmap, icon, or cursor from a resource (.res) file.

Syntax

LoadResPicture(*index*, *format***)**

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

Part	Description
index	Required. Integer or string specifying the identifier (ID) of the data in the resource file. The resource whose ID is 1 is reserved for the application icon.
format	Required. Value or constant that specifies the format of the data being returned, as described in Settings.

Settings

The settings for *format* are:

Constant	Value	Description
vbResBitmap	0	Bitmap resource
vbResIcon	1	Icon resource
vbResCursor	2	Cursor resource

Remarks

You can use the **LoadResPicture** function instead of referring to graphics stored in the **Picture** property of a **Form** or controls.

Storing bitmaps, icons, or cursors in and accessing them from resource files improves load time because you can load them individually as needed from the resource file, rather than all at once when a **Form** is loaded.

Using **LoadResPicture** is useful for localizing a Visual Basic application because the resources that need to be translated are isolated in one resource file and there is no need to access the source code or recompile the application.

Visual Basic Reference

Visual Studio 6.0

LoadResString Function

See Also Example Applies To

Loads a string from a resource (.res) file.

Syntax

LoadResString(*index***)**

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

Part	Description
index	Required. Integer specifying the identifier (ID) of the data in the resource file. The resource whose ID is 1 is reserved for the application icon.

Remarks

You can use the **LoadResString** function instead of string literals in your code. Storing long strings of data in and accessing them from resource files improves load time because you can load them individually as needed from the resource file, rather than all at once when a form is loaded.

Using **LoadResString** is useful for localizing a Visual Basic application because the resources that need to be translated are isolated in one resource file and there is no need to access the source code or recompile the application.

Visual Basic for Applications Reference

Visual Studio 6.0

Loc Function

See Also Example Specifics

Returns a Long specifying the current read/write position within an open file.

Syntax

Loc(*filenumber***)**

The required *filenumber* argument is any valid Integer file number.

Remarks

The following describes the return value for each file access mode:

Mode	Return Value
Random	Number of the last record read from or written to the file.
Sequential	Current byte position in the file divided by 128. However, information returned by **Loc** for sequential files is neither used nor required.
Binary	Position of the last byte read or written.

Visual Basic for Applications Reference

Loc Function Example

This example uses the **Loc** function to return the current read/write position within an open file. This example assumes that `TESTFILE` is a text file with a few lines of sample data.

```
Dim MyLocation, MyLine
Open "TESTFILE" For Binary As #1   ' Open file just created.
Do While MyLocation < LOF(1)   ' Loop until end of file.
   MyLine = MyLine & Input(1, #1)   ' Read character into variable.
   MyLocation = Loc(1)   ' Get current position within file.
' Print to the Immediate window.
   Debug.Print MyLine; Tab; MyLocation
Loop
Close #1   ' Close file.
```

Visual Basic for Applications Reference

Visual Studio 6.0

LOF Function

See Also Example Specifics

Returns a Long representing the size, in bytes, of a file opened using the **Open** statement.

Syntax

LOF(*filenumber***)**

The required *filenumber* argument is an Integer containing a valid file number.

Note Use the **FileLen** function to obtain the length of a file that is not open.

Visual Basic for Applications Reference

LOF Function Example

This example uses the **LOF** function to determine the size of an open file. This example assumes that `TESTFILE` is a text file containing sample data.

```
Dim FileLength
Open "TESTFILE" For Input As #1    ' Open file.
FileLength = LOF(1)    ' Get length of file.
Close #1    ' Close file.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Log Function

See Also Example Specifics

Returns a **Double** specifying the natural logarithm of a number.

Syntax

Log(*number***)**

The required *number* argument is a Double or any valid numeric expression greater than zero.

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:

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

Visual Basic for Applications Reference

Log Function Example

This example uses the **Log** function to return the natural logarithm of a number.

```
Dim MyAngle, MyLog
' Define angle in radians.
MyAngle = 1.3
' Calculate inverse hyperbolic sine.
MyLog = Log(MyAngle + Sqr(MyAngle * MyAngle + 1))
```

Visual Basic for Applications Reference

Visual Studio 6.0

LTrim, RTrim, and Trim Functions

See Also Example Specifics

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

Syntax

LTrim(*string*)

RTrim(*string*)

Trim(*string*)

The required *string* argument is any valid string expression. If *string* contains Null, **Null** is returned.

Visual Basic for Applications Reference

LTrim, RTrim, and Trim Functions Example

This example uses the **LTrim** function to strip leading spaces and the **RTrim** function to strip trailing spaces from a string variable. It uses the **Trim** function to strip both types of spaces.

```
Dim MyString, TrimString
MyString = "  <-Trim->  "   ' Initialize string.
TrimString = LTrim(MyString)   ' TrimString = "<-Trim->  ".
TrimString = RTrim(MyString)   ' TrimString = "  <-Trim->".
TrimString = LTrim(RTrim(MyString))   ' TrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
TrimString = Trim(MyString)   ' TrimString = "<-Trim->".
```