Visual Basic for Applications Reference

Visual Studio 6.0

Second Function

See Also Example Specifics

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 59, inclusive, representing the second of the minute.

Syntax

Second(*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

Second Function Example

This example uses the **Second** function to obtain the second of the minute 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, MySecond
MyTime = #4:35:17 PM#   ' Assign a time.
MySecond = Second(MyTime)   ' MySecond contains 17.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Seek Function

See Also Example Specifics

Returns a Long specifying the current read/write position within a file opened using the **Open** statement.

Syntax

Seek(*filenumber***)**

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

Remarks

Seek returns a value between 1 and 2,147,483,647 (equivalent to 2^31 1), inclusive.

The following describes the return values for each file access mode.

Mode	Return Value
Random	Number of the next record read or written
Binary, **Output**, **Append**, **Input**	Byte position at which the next operation takes place. The first byte in a file is at position 1, the second byte is at position 2, and so on.

Visual Basic for Applications Reference

Seek Function Example

This example uses the **Seek** function to return the current file position. The example assumes `TESTFILE` is a file containing records of the user-defined type `Record`.

```
Type Record   ' Define user-defined type.
   ID As Integer
   Name As String * 20
End Type
```

For files opened in Random mode, **Seek** returns number of next record.

```
Dim MyRecord As Record   ' Declare variable.
Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
Do While Not EOF(1)   ' Loop until end of file.
   Get #1, , MyRecord   ' Read next record.
   Debug.Print Seek(1)   ' Print record number to the
         ' Immediate window.
Loop
Close #1   ' Close file.
```

For files opened in modes other than Random mode, **Seek** returns the byte position at which the next operation takes place. Assume `TESTFILE` is a file containing a few lines of text.

```
Dim MyChar
Open "TESTFILE" For Input As #1   ' Open file for reading.
Do While Not EOF(1)   ' Loop until end of file.
   MyChar = Input(1, #1)   ' Read next character of data.
   Debug.Print Seek(1)   ' Print byte position to the
         ' Immediate window.
Loop
Close #1   ' Close file.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Sgn Function

See Also Example Specifics

Returns a **Variant** (**Integer**) indicating the sign of a number.

Syntax

Sgn(*number***)**

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

Return Values

If *number* is	**Sgn returns**
Greater than zero	1
Equal to zero	0
Less than zero	-1

Remarks

The sign of the *number* argument determines the return value of the **Sgn** function.

Visual Basic for Applications Reference

Sgn Function Example

This example uses the **Sgn** function to determine the sign of a number.

```
Dim MyVar1, MyVar2, MyVar3, MySign
MyVar1 = 12: MyVar2 = -2.4: MyVar3 = 0
MySign = Sgn(MyVar1)   ' Returns 1.
MySign = Sgn(MyVar2)   ' Returns -1.
MySign = Sgn(MyVar3)   ' Returns 0.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Shell Function

See Also Example Specifics

Runs an executable program and returns a **Variant** (**Double**) representing the program's task ID if successful, otherwise it returns zero.

Syntax

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

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

Part	Description
pathname	Required; **Variant** (**String**). Name of the program to execute and any required arguments or command-line switches; may include directory or folder and drive.
windowstyle	Optional. **Variant** (**Integer**) corresponding to the style of the window in which the program is to be run. If ***windowstyle*** is omitted, the program is started minimized with focus.

The ***windowstyle*** named argument has these values:

Constant	Value	Description
vbHide	0	Window is hidden and focus is passed to the hidden window.
vbNormalFocus	1	Window has focus and is restored to its original size and position.
vbMinimizedFocus	2	Window is displayed as an icon with focus.
vbMaximizedFocus	3	Window is maximized with focus.
vbNormalNoFocus	4	Window is restored to its most recent size and position. The currently active window remains active.
vbMinimizedNoFocus	6	Window is displayed as an icon. The currently active window remains active.

Remarks

If the **Shell** function successfully executes the named file, it returns the task ID of the started program. The task ID is a unique number that identifies the running program. If the **Shell** function can't start the named program, an error occurs.

Note By default, the **Shell** function runs other programs asynchronously. This means that a program started with **Shell** might not finish executing before the statements following the **Shell** function are executed.

> **Security Note** If you do not enclose the path and file specification in quotes, there is a security risk if the file name or a path node contains spaces. If the path node specification is not inside quotes, for example `\Program Files` and a program named `Program.exe` had been installed in C:\, for example by illicit tampering, Windows would execute it instead of `MyFile.exe`.

Visual Basic for Applications Reference

Shell Function Example

This example uses the **Shell** function to run an application specified by the user.

```
' Specifying 1 as the second argument opens the application in
' normal size and gives it the focus.
Dim RetVal
RetVal = Shell("C:\WINDOWS\CALC.EXE", 1)   ' Run Calculator.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Sin Function

See Also Example Specifics

Returns a **Double** specifying the sine of an angle.

Syntax

Sin(*number***)**

The required *number* argument is a Double or any valid numeric expression that expresses an angle in radians.

Remarks

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

The result lies in the range -1 to 1.

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

Visual Basic for Applications Reference

Sin Function Example

This example uses the **Sin** function to return the sine of an angle.

```
Dim MyAngle, MyCosecant
MyAngle = 1.3   ' Define angle in radians.
MyCosecant = 1 / Sin(MyAngle)   ' Calculate cosecant.
```

Visual Basic for Applications Reference

Visual Studio 6.0

SLN Function

See Also Example Specifics

Returns a Double specifying the straight-line depreciation of an asset for a single period.

Syntax

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

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

Part	Description
cost	Required. **Double** specifying initial cost of the asset.
salvage	Required. **Double** specifying value of the asset at the end of its useful life.
life	Required. **Double** specifying length of the useful life of the asset.

Remarks

The depreciation period must be expressed in the same unit as the ***life*** argument. All arguments must be positive numbers.

Visual Basic for Applications Reference

SLN Function Example

This example uses the **SLN** function to return the straight-line depreciation of an asset for a single period given the asset's initial cost (`InitCost`), the salvage value at the end of the asset's useful life (`SalvageVal`), and the total life of the asset in years (`LifeTime`).

```
Dim Fmt, InitCost, SalvageVal, MonthLife, LifeTime, PDepr
Const YEARMONTHS = 12   ' Number of months in a year.
Fmt = "###,##0.00"   ' Define money format.
InitCost = InputBox("What's the initial cost of the asset?")
SalvageVal = InputBox("What's the asset's value at the end of its useful life?")
MonthLife = InputBox("What's the asset's useful life in months?")
Do While MonthLife < YEARMONTHS   ' Ensure period is >= 1 year.
   MsgBox "Asset life must be a year or more."
   MonthLife = InputBox("What's the asset's useful life in months?")
Loop
LifeTime = MonthLife / YEARMONTHS   ' Convert months to years.
If LifeTime <> Int(MonthLife / YEARMONTHS) Then
   LifeTime = Int(LifeTime + 1)   ' Round up to nearest year.
End If
PDepr = SLN(InitCost, SalvageVal, LifeTime)
MsgBox "The depreciation is " & Format(PDepr, Fmt) & " per year."
```

Visual Basic for Applications Reference

Visual Studio 6.0

Space Function

See Also Example Specifics

Returns a **Variant** (**String**) consisting of the specified number of spaces.

Syntax

Space(*number*)

The required *number* argument is the number of spaces you want in the string.

Remarks

The **Space** function is useful for formatting output and clearing data in fixed-length strings.

Visual Basic for Applications Reference

Space Function Example

This example uses the **Space** function to return a string consisting of a specified number of spaces.

```
Dim MyString
' Returns a string with 10 spaces.
MyString = Space(10)

' Insert 10 spaces between two strings.
MyString = "Hello" & Space(10) & "World"
```

Visual Basic for Applications Reference

Visual Studio 6.0

Spc Function

See Also Example Specifics

Used with the **Print #** statement or the **Print** method to position output.

Syntax

Spc(*n*)

The required *n* argument is the number of spaces to insert before displaying or printing the next expression in a list.

Remarks

If *n* is less than the output line width, the next print position immediately follows the number of spaces printed. If *n* is greater than the output line width, **Spc** calculates the next print position using the formula:

currentprintposition + (*n* **Mod** *width*)

For example, if the current print position is 24, the output line width is 80, and you specify **Spc(**90**)**, the next print will start at position 34 (current print position + the remainder of 90/80). If the difference between the current print position and the output line width is less than *n* (or *n* **Mod** *width*), the **Spc** function skips to the beginning of the next line and generates spaces equal to *n* (*width currentprintposition*).

Note Make sure your tabular columns are wide enough to accommodate wide letters.

When you use the **Print** method with a proportionally spaced font, the width of space characters printed using the **Spc** function is always an average of the width of all characters in the point size for the chosen font. However, there is no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, the uppercase letter W occupies more than one fixed-width column and the lowercase letter i occupies less than one fixed-width column.

Visual Basic for Applications Reference

Spc Function Example

This example uses the **Spc** function to position output in a file and in the **Immediate** window.

```
' The Spc function can be used with the Print # statement.
Open "TESTFILE" For Output As #1   ' Open file for output.
Print #1, "10 spaces between here"; Spc(10); "and here."
Close #1   ' Close file.
```

The following statement causes the text to be printed in the **Immediate** window (using the **Print** method), preceded by 30 spaces.

```
Debug.Print Spc(30); "Thirty spaces later..."
```

This documentation is archived and is not being maintained.

Visual Basic for Applications Reference

Visual Studio 6.0

Split Function

See Also Example Specifics

Description

Returns a zero-based, one-dimensional array containing a specified number of substrings.

Syntax

Split(*expression*[, *delimiter*[, *limit*[, *compare*]]])

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

Part	Description
expression	Required. String expression containing substrings and delimiters. If *expression* is a zero-length string(""), **Split** returns an empty array, that is, an array with no elements and no data.
delimiter	Optional. String character used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If ***delimiter*** is a zero-length string, a single-element array containing the entire ***expression*** string is returned.
limit	Optional. Number of substrings to be returned; 1 indicates that all substrings are returned.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values.

Settings

The ***compare*** argument can have the following values:

Constant	Value	Description
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.

Visual Basic for Applications Reference

Visual Studio 6.0

Sqr Function

See Also Example Specifics

Returns a **Double** specifying the square root of a number.

Syntax

Sqr(*number***)**

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

Visual Basic for Applications Reference

Sqr Function Example

This example uses the **Sqr** function to calculate the square root of a number.

```
Dim MySqr
MySqr = Sqr(4)   ' Returns 2.
MySqr = Sqr(23)   ' Returns 4.79583152331272.
MySqr = Sqr(0)   ' Returns 0.
MySqr = Sqr(-4)   ' Generates a run-time error.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Str Function

See Also Example Specifics

Returns a **Variant** (**String**) representation of a number.

Syntax

Str(*number*)

The required *number* argument is a Long containing any valid numeric expression.

Remarks

When numbers are converted to strings, a leading space is always reserved for the sign of *number*. If *number* is positive, the returned string contains a leading space and the plus sign is implied.

Use the **Format** function to convert numeric values you want formatted as dates, times, or currency or in other user-defined formats. Unlike **Str**, the **Format** function doesn't include a leading space for the sign of *number*.

Note The **Str** function recognizes only the period (**.**) as a valid decimal separator. When different decimal separators may be used (for example, in international applications), use **CStr** to convert a number to a string.

Visual Basic for Applications Reference

Str Function Example

This example uses the **Str** function to return a string representation of a number. When a number is converted to a string, a leading space is always reserved for its sign.

```
Dim MyString
MyString = Str(459)   ' Returns " 459".
MyString = Str(-459.65)   ' Returns "-459.65".
MyString = Str(459.001)   ' Returns " 459.001".
```

Visual Basic for Applications Reference

Visual Studio 6.0

StrComp Function

See Also Example Specifics

Returns a **Variant** (**Integer**) indicating the result of a string comparison.

Syntax

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

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

Part	Description
string1	Required. Any valid string expression.
string2	Required. Any valid string expression.
compare	Optional. Specifies the type of string comparison. If the ***compare*** argument is Null, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of 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

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

If	StrComp returns
string1 is less than ***string2***	-1
string1 is equal to ***string2***	0
string1 is greater than ***string2***	1
string1 or ***string2*** is **Null**	**Null**

Visual Basic for Applications Reference

StrComp Function Example

This example uses the **StrComp** function to return the results of a string comparison. If the third argument is 1, a textual comparison is performed; if the third argument is 0 or omitted, a binary comparison is performed.

```
Dim MyStr1, MyStr2, MyComp
MyStr1 = "ABCD": MyStr2 = "abcd"   ' Define variables.
MyComp = StrComp(MyStr1, MyStr2, 1)   ' Returns 0.
MyComp = StrComp(MyStr1, MyStr2, 0)   ' Returns -1.
MyComp = StrComp(MyStr2, MyStr1)   ' Returns 1.
```

Visual Basic for Applications Reference

Visual Studio 6.0

StrConv Function

See Also Example Specifics

Returns a **Variant** (**String**) converted as specified.

Syntax

StrConv(***string, conversion, LCID***)

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

Part	Description
string	Required. String expression to be converted.
conversion	Required. Integer. The sum of values specifying the type of conversion to perform.
LCID	Optional. The LocaleID, if different than the system LocaleID. (The system LocaleID is the default.)

Settings

The ***conversion*** argument settings are:

Constant	Value	Description
vbUpperCase	1	Converts the string to uppercase characters.
vbLowerCase	2	Converts the string to lowercase characters.
vbProperCase	3	Converts the first letter of every word in string to uppercase.
vbWide*	4*	Converts narrow (single-byte) characters in string to wide (double-byte) characters.
vbNarrow*	8*	Converts wide (double-byte) characters in string to narrow (single-byte) characters.
vbKatakana**	16**	Converts Hiragana characters in string to Katakana characters.
vbHiragana**	32**	Converts Katakana characters in string to Hiragana characters.
vbUnicode	64	Converts the string to Unicode using the default code page of the system.

vbFromUnicode	128	Converts the string from Unicode to the default code page of the system.

*Applies to Far East locales.

**Applies to Japan only.

Note These constants are specified by Visual Basic for Applications. As a result, they may be used anywhere in your code in place of the actual values. Most can be combined, for example, **vbUpperCase + vbWide**, except when they are mutually exclusive, for example, **vbUnicode + vbFromUnicode**. The constants **vbWide**, **vbNarrow**, **vbKatakana**, and **vbHiragana** cause run-time errors when used in locales where they do not apply.

The following are valid word separators for proper casing: Null (**Chr$(**0**)**), horizontal tab (**Chr$(**9**)**), linefeed (**Chr$(**10**)**), vertical tab (**Chr$(**11**)**), form feed (**Chr$(**12**)**), carriage return (**Chr$(**13**)**), space (SBCS) (**Chr$(**32**)**). The actual value for a space varies by country for DBCS.

Remarks

When you're converting from a **Byte** array in ANSI format to a string, you should use the **StrConv** function. When you're converting from such an array in Unicode format, use an assignment statement.

Visual Basic for Applications Reference

Visual Studio 6.0

String Function

See Also Example Specifics

Returns a **Variant** (**String**) containing a repeating character string of the length specified.

Syntax

String(***number***, ***character***)

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

Part	Description
number	Required; Long. Length of the returned string. If ***number*** contains Null, **Null** is returned.
character	Required; Variant. Character code specifying the character or string expression whose first character is used to build the return string. If ***character*** contains **Null**, **Null** is returned.

Remarks

If you specify a number for ***character*** greater than 255, **String** converts the number to a valid character code using the formula:

character **Mod** 256

Visual Basic for Applications Reference

String Function Example

This example uses the **String** function to return repeating character strings of the length specified.

```
Dim MyString
MyString = String(5, "*")   ' Returns "*****".
MyString = String(5, 42)   ' Returns "*****".
MyString = String(10, "ABC")   ' Returns "AAAAAAAAAA".
```

Visual Basic for Applications Reference

Visual Studio 6.0

StrReverse Function

See Also Example Specifics

Description

Returns a string in which the character order of a specified string is reversed.

Syntax

StrReverse(*expression*)

The ***expression*** argument is the string whose characters are to be reversed. If ***expression*** is a zero-length string (""), a zero-length string is returned. If ***expression*** is **Null**, an error occurs.

Visual Basic for Applications Reference

Visual Studio 6.0

Switch Function

See Also Example Specifics

Evaluates a list of expressions and returns a **Variant** value or an expression associated with the first expression in the list that is **True**.

Syntax

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

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

Part	Description
expr	Required. Variant expression you want to evaluate.
value	Required. Value or expression to be returned if the corresponding expression is **True**.

Remarks

The **Switch** function argument list consists of pairs of expressions and values. The expressions are evaluated from left to right, and the value associated with the first expression to evaluate to **True** is returned. If the parts aren't properly paired, a run-time error occurs. For example, if *expr-1* is **True**, **Switch** returns *value-1*. If *expr-1* is **False**, but *expr-2* is **True**, **Switch** returns *value-2*, and so on.

Switch returns a Null value if:

- None of the expressions is **True**.
- The first **True** expression has a corresponding value that is **Null**.

Switch evaluates all of the expressions, even though it returns only one of them. For this reason, you should watch for undesirable side effects. For example, if the evaluation of any expression results in a division by zero error, an error occurs.

Visual Basic for Applications Reference

Switch Function Example

This example uses the **Switch** function to return the name of a language that matches the name of a city.

```
Function MatchUp (CityName As String)
   Matchup = Switch(CityName = "London", "English", CityName _
               = "Rome", "Italian", CityName = "Paris", "French")
End Function
```

Visual Basic for Applications Reference

Visual Studio 6.0

SYD Function

See Also Example Specifics

Returns a Double specifying the sum-of-years' digits depreciation of an asset for a specified period.

Syntax

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

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

Part	Description
cost	Required. **Double** specifying initial cost of the asset.
salvage	Required. **Double** specifying value of the asset at the end of its useful life.
life	Required. **Double** specifying length of the useful life of the asset.
period	Required. **Double** specifying period for which asset depreciation is calculated.

Remarks

The ***life*** and ***period*** arguments must be expressed in the same units. For example, if ***life*** is given in months, ***period*** must also be given in months. All arguments must be positive numbers.

Visual Basic for Applications Reference

SYD Function Example

This example uses the **SYD** function to return the depreciation of an asset for a specified period given the asset's initial cost (`InitCost`), the salvage value at the end of the asset's useful life (`SalvageVal`), and the total life of the asset in years (`LifeTime`). The period in years for which the depreciation is calculated is `PDepr`.

```
Dim Fmt, InitCost, SalvageVal, MonthLife, LifeTime, DepYear, PDepr
Const YEARMONTHS = 12   ' Number of months in a year.
Fmt = "###,##0.00"   ' Define money format.
InitCost = InputBox("What's the initial cost of the asset?")
SalvageVal = InputBox("What's the asset's value at the end of its life?")
MonthLife = InputBox("What's the asset's useful life in months?")
Do While MonthLife < YEARMONTHS   ' Ensure period is >= 1 year.
   MsgBox "Asset life must be a year or more."
   MonthLife = InputBox("What's the asset's useful life in months?")
Loop
LifeTime = MonthLife / YEARMONTHS   ' Convert months to years.
If LifeTime <> Int(MonthLife / YEARMONTHS) Then
   LifeTime = Int(LifeTime + 1)   ' Round up to nearest year.
End If
DepYear = CInt(InputBox("For which year do you want depreciation?"))
Do While DepYear < 1 Or DepYear > LifeTime
   MsgBox "You must enter at least 1 but not more than " & LifeTime
   DepYear = CInt(InputBox("For what year do you want depreciation?"))
Loop
PDepr = SYD(InitCost, SalvageVal, LifeTime, DepYear)
MsgBox "The depreciation for year " & DepYear & " is " & Format(PDepr, Fmt) & "."
```

Visual Basic for Applications Reference

Visual Studio 6.0

Tab Function

See Also Example Specifics

Used with the **Print #** statement or the **Print** method to position output.

Syntax

Tab[(*n*)]

The optional *n* argument is the column number moved to before displaying or printing the next expression in a list. If omitted, **Tab** moves the insertion point to the beginning of the next print zone. This allows **Tab** to be used instead of a comma in locales where the comma is used as a decimal separator.

Remarks

If the current print position on the current line is greater than *n*, **Tab** skips to the *n*th column on the next output line. If *n* is less than 1, **Tab** moves the print position to column 1. If *n* is greater than the output line width, **Tab** calculates the next print position using the formula:

n **Mod** *width*

For example, if *width* is 80 and you specify **Tab(**90**)**, the next print will start at column 10 (the remainder of 90/80). If *n* is less than the current print position, printing begins on the next line at the calculated print position. If the calculated print position is greater than the current print position, printing begins at the calculated print position on the same line.

The leftmost print position on an output line is always 1. When you use the **Print #** statement to print to files, the rightmost print position is the current width of the output file, which you can set using the **Width #** statement.

Note Make sure your tabular columns are wide enough to accommodate wide letters.

When you use the **Tab** function with the **Print** method, the print surface is divided into uniform, fixed-width columns. The width of each column is an average of the width of all characters in the point size for the chosen font. However, there is no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, the uppercase letter W occupies more than one fixed-width column and the lowercase letter i occupies less than one fixed-width column.

Visual Basic for Applications Reference

Tab Function Example

This example uses the **Tab** function to position output in a file and in the **Immediate** window.

```
' The Tab function can be used with the Print # statement.
Open "TESTFILE" For Output As #1   ' Open file for output.
' The second word prints at column 20.
Print #1, "Hello"; Tab(20); "World."
' If the argument is omitted, cursor is moved to the next print zone.
Print #1, "Hello"; Tab; "World"
Close #1   ' Close file.
```

The **Tab** function can also be used with the **Print** method. The following statement prints text starting at column 10.

```
Debug.Print Tab(10); "10 columns from start."
```

Visual Basic for Applications Reference

Visual Studio 6.0

Tan Function

See Also Example Specifics

Returns a **Double** specifying the tangent of an angle.

Syntax

Tan(*number***)**

The required *number* argument is a Double or any valid numeric expression that expresses an angle in radians.

Remarks

Tan takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

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

Visual Basic for Applications Reference

Tan Function Example

This example uses the **Tan** function to return the tangent of an angle.

```
Dim MyAngle, MyCotangent
MyAngle = 1.3   ' Define angle in radians.
MyCotangent = 1 / Tan(MyAngle)   ' Calculate cotangent.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Time Function

See Also Example Specifics

Returns a **Variant** (**Date**) indicating the current system time.

Syntax

Time

Remarks

To set the system time, use the **Time** statement.

Visual Basic for Applications Reference

Time Function Example

This example uses the **Time** function to return the current system time.

```
Dim MyTime
MyTime = Time   ' Return current system time.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Timer Function

See Also Example Specifics

Returns a **Single** representing the number of seconds elapsed since midnight.

Syntax

Timer

Remarks

In Microsoft Windows the **Timer** function returns fractional portions of a second.

Visual Basic for Applications Reference

Timer Function Example

This example uses the **Timer** function to pause the application. The example also uses **DoEvents** to yield to other processes during the pause.

```
Dim PauseTime, Start, Finish, TotalTime
If (MsgBox("Press Yes to pause for 5 seconds", 4)) = vbYes Then
   PauseTime = 5   ' Set duration.
   Start = Timer   ' Set start time.
   Do While Timer < Start + PauseTime
      DoEvents   ' Yield to other processes.
   Loop
   Finish = Timer   ' Set end time.
   TotalTime = Finish - Start   ' Calculate total time.
   MsgBox "Paused for " & TotalTime & " seconds"
Else
   End
End If
```

Visual Basic for Applications Reference

Visual Studio 6.0

TimeSerial Function

See Also Example Specifics

Returns a **Variant** (**Date**) containing the time for a specific hour, minute, and second.

Syntax

TimeSerial(*hour, minute, second*)

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

Part	Description
hour	Required; **Variant** (**Integer**). Number between 0 (12:00 A.M.) and 23 (11:00 P.M.), inclusive, or a numeric expression.
minute	Required; **Variant** (**Integer**). Any numeric expression.
second	Required; **Variant** (**Integer**). Any numeric expression.

Remarks

To specify a time, such as 11:59:59, the range of numbers for each **TimeSerial** argument should be in the normal range for the unit; that is, 023 for hours and 059 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time. The following example uses expressions instead of absolute time numbers. The **TimeSerial** function returns a time for 15 minutes before (`-15`) six hours before noon (`12 - 6`), or 5:45:00 A.M.

```
TimeSerial(12 - 6, -15, 0)
```

When any argument exceeds the normal range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 75 minutes, it is evaluated as one hour and 15 minutes. If any single argument is outside the range -32,768 to 32,767, an error occurs. If the time specified by the three arguments causes the date to fall outside the acceptable range of dates, an error occurs.

Visual Basic for Applications Reference

TimeSerial Function Example

This example uses the **TimeSerial** function to return a time for the specified hour, minute, and second.

```
Dim MyTime
MyTime = TimeSerial(16, 35, 17)   ' MyTime contains serial
   ' representation of 4:35:17 PM.
```

Visual Basic for Applications Reference

Visual Studio 6.0

TimeValue Function

See Also Example Specifics

Returns a **Variant** (**Date**) containing the time.

Syntax

TimeValue(*time***)**

The required *time* argument is normally a string expression representing a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive. However, *time* can also be any expression that represents a time in that range. If *time* contains Null, **Null** is returned.

Remarks

You can enter valid times using a 12-hour or 24-hour clock. For example, `"2:24PM"` and `"14:24"` are both valid *time* arguments.

If the *time* argument contains date information, **TimeValue** doesn't return it. However, if *time* includes invalid date information, an error occurs.

Visual Basic for Applications Reference

TimeValue Function Example

This example uses the **TimeValue** function to convert a string to a time. You can also use date literals to directly assign a time to a **Variant** or **Date** variable, for example, MyTime = #4:35:17 PM#.

```
Dim MyTime
MyTime = TimeValue("4:35:17 PM")   ' Return a time.
```

Visual Basic for Applications Reference

Visual Studio 6.0

TypeName Function

See Also Example Specifics

Returns a **String** that provides information about a variable.

Syntax

TypeName(*varname***)**

The required *varname* argument is a Variant containing any variable except a variable of a user-defined type.

Remarks

The string returned by **TypeName** can be any one of the following:

String returned	Variable
object type	An object whose type is *objecttype*
Byte	Byte value
Integer	Integer
Long	Long integer
Single	Single-precision floating-point number
Double	Double-precision floating-point number
Currency	Currency value
Decimal	Decimal value
Date	Date value
String	String
Boolean	Boolean value
Error	An error value
Empty	Uninitialized

Null	No valid data
Object	An object
Unknown	An object whose type is unknown
Nothing	Object variable that doesn't refer to an object

If *varname* is an array, the returned string can be any one of the possible returned strings (or **Variant**) with empty parentheses appended. For example, if *varname* is an array of integers, **TypeName** returns "`Integer()`".

TypeName 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 displays the Visual Basic object type of the selection. You can run this example with cells selected, with a single oval selected, or with several different graphic objects selected.

```
Worksheets("Sheet1").Activate
MsgBox "The selection object type is " & TypeName(Selection)
```

Visual Basic for Applications Reference

Visual Studio 6.0

UBound Function

See Also Example Specifics

Returns a Long containing the largest available subscript for the indicated dimension of an array.

Syntax

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

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

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

Remarks

The **UBound** function is used with the **LBound** function to determine the size of an array. Use the **LBound** function to find the lower limit of an array dimension.

UBound returns the following values for an array with these dimensions:

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

Statement	Return Value
`UBound(A, 1)`	100
`UBound(A, 2)`	3
`UBound(A, 3)`	4

UBound 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. 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

UCase Function

See Also Example Specifics

Returns a **Variant** (**String**) containing the specified string, converted to uppercase.

Syntax

UCase(*string*)

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

Remarks

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

Visual Basic for Applications Reference

UCase Function Example

This example uses the **UCase** function to return an uppercase version of a string.

```
Dim LowerCase, UpperCase
LowerCase = "Hello World 1234"   ' String to convert.
UpperCase = UCase(LowerCase)   ' Returns "HELLO WORLD 1234".
```

Visual Basic for Applications Reference

Visual Studio 6.0

Val Function

See Also Example Specifics

Returns the numbers contained in a string as a numeric value of appropriate type.

Syntax

Val(*string***)**

The required *string* argument is any valid string expression.

Remarks

The **Val** function stops reading the string at the first character it can't recognize as part of a number. Symbols and characters that are often considered parts of numeric values, such as dollar signs and commas, are not recognized. However, the function recognizes the radix prefixes &O (for octal) and &H (for hexadecimal). Blanks, tabs, and linefeed characters are stripped from the argument.

The following returns the value 1615198:

```
Val("    1615 198th Street N.E.")
```

In the code below, **Val** returns the decimal value -1 for the hexadecimal value shown:

```
Val("&HFFFF")
```

Note The **Val** function recognizes only the period (.) as a valid decimal separator. When different decimal separators are used, as in international applications, use **CDbl** instead to convert a string to a number.

Visual Basic for Applications Reference

Val Function Example

This example uses the **Val** function to return the numbers contained in a string.

```
Dim MyValue
MyValue = Val("2457")   ' Returns 2457.
MyValue = Val(" 2 45 7")   ' Returns 2457.
MyValue = Val("24 and 57")   ' Returns 24.
```

Visual Basic for Applications Reference

Visual Studio 6.0

VarType Function

See Also Example Specifics

Returns an **Integer** indicating the subtype of a variable.

Syntax

VarType(*varname***)**

The required *varname* argument is a Variant containing any variable except a variable of a user-defined type.

Return Values

Constant	Value	Description
vbEmpty	0	Empty (uninitialized)
vbNull	1	Null (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency value
vbDate	7	Date value
vbString	8	String
vbObject	9	Object
vbError	10	Error value
vbBoolean	11	Boolean value
vbVariant	12	**Variant** (used only with arrays of variants)
vbDataObject	13	A data access object

vbDecimal	14	Decimal value
vbByte	17	Byte value
vbUserDefinedType	36	Variants that contain user-defined types
vbArray	8192	Array

Note These constants are specified by Visual Basic for Applications. The names can be used anywhere in your code in place of the actual values.

Remarks

The **VarType** function never returns the value for **vbArray** by itself. It is always added to some other value to indicate an array of a particular type. The constant **vbVariant** is only returned in conjunction with **vbArray** to indicate that the argument to the **VarType** function is an array of type **Variant**. For example, the value returned for an array of integers is calculated as **vbInteger** + **vbArray**, or 8194. If an object has a default property, **VarType (***object***)** returns the type of the object's default property.

Visual Basic for Applications Reference

VarType Function Example

This example uses the **VarType** function to determine the subtype of a variable.

```
Dim IntVar, StrVar, DateVar, MyCheck
' Initialize variables.
IntVar = 459: StrVar = "Hello World": DateVar = #2/12/69#
MyCheck = VarType(IntVar)   ' Returns 2.
MyCheck = VarType(DateVar)   ' Returns 7.
MyCheck = VarType(StrVar)   ' Returns 8.
```

Visual Basic for Applications Reference

Visual Studio 6.0

Weekday Function

See Also Example Specifics

Returns a **Variant** (**Integer**) containing a whole number representing the day of the week.

Syntax

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

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

Part	Description
date	Required. Variant, numeric expression, string expression, or any combination, that can represent a date. If ***date*** contains Null, **Null** is returned.
firstdayofweek	Optional. A constant that specifies the first day of the week. If not specified, **vbSunday** is assumed.

Settings

The ***firstdayofweek*** argument has these settings:

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

Return Values

The **Weekday** function can return any of these values:

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Remarks

If the **Calendar** property setting is Gregorian, the returned integer represents the Gregorian day of the week for the date argument. If the calendar is Hijri, the returned integer represents the Hijri day of the week for the date argument. For Hijri dates, the argument number is any numeric expression that can represent a date and/or time from 1/1/100 (Gregorian Aug 2, 718) through 4/3/9666 (Gregorian Dec 31, 9999).

Visual Basic for Applications Reference

Weekday Function Example

This example uses the **Weekday** function to obtain the day of the week from a specified date.

```
Dim MyDate, MyWeekDay
MyDate = #February 12, 1969#   ' Assign a date.
MyWeekDay = Weekday(MyDate)   ' MyWeekDay contains 4 because
   ' MyDate represents a Wednesday.
```

Visual Basic for Applications Reference

Visual Studio 6.0

WeekdayName Function

See Also Example Specifics

Description

Returns a string indicating the specified day of the week.

Syntax

WeekdayName(*weekday***, *abbreviate*, *firstdayofweek*)**

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

Part	Description
weekday	Required. The numeric designation for the day of the week. Numeric value of each day depends on setting of the *firstdayofweek* setting.
abbreviate	Optional. **Boolean** value that indicates if the weekday name is to be abbreviated. If omitted, the default is **False**, which means that the weekday name is not abbreviated.
firstdayofweek	Optional. Numeric value indicating the first day of the week. See Settings section for values.

Settings

The *firstdayofweek* argument can have the following values:

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

vbFriday	6	Friday
vbSaturday	7	Saturday

Visual Basic for Applications Reference

Visual Studio 6.0

Year Function

See Also Example Specifics

Returns a **Variant** (**Integer**) containing a whole number representing the year.

Syntax

Year(*date***)**

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

Note If the **Calendar** property setting is Gregorian, the returned integer represents the Gregorian year for the date argument. If the calendar is Hijri, the returned integer represents the Hijri year for the date argument. For Hijri dates, the argument number is any numeric expression that can represent a date and/or time from 1/1/100 (Gregorian Aug 2, 718) through 4/3/9666 (Gregorian Dec 31, 9999).

Visual Basic for Applications Reference

Year Function Example

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

```
Dim MyDate, MyYear
MyDate = #February 12, 1969#   ' Assign a date.
MyYear = Year(MyDate)   ' MyYear contains 1969.
```