VBScript Reference and Samples

Date/Time Functions

Function	Description
CDate	Converts a valid date and time expression to the variant of subtype Date
Date	Returns the current system date
DateAdd	Returns a date to which a specified time interval has been added
DateDiff	Returns the number of intervals between two dates
DatePart	Returns the specified part of a given date
DateSerial	Returns the date for a specified year, month, and day
DateValue	Returns a date
Day	Returns a number that represents the day of the month (between 1 and 31, inclusive)
FormatDateTime	Returns an expression formatted as a date or time
Hour	Returns a number that represents the hour of the day (between 0 and 23, inclusive)
IsDate	Returns a Boolean value that indicates if the evaluated expression can be converted to a date
Minute	Returns a number that represents the minute of the hour (between 0 and 59, inclusive)
Month	Returns a number that represents the month of the year (between 1 and 12, inclusive)
MonthName	Returns the name of a specified month
Now	Returns the current system date and time
Second	Returns a number that represents the second of the minute (between 0 and 59, inclusive)
Time	Returns the current system time
Timer	Returns the number of seconds since 12:00 AM
TimeSerial	Returns the time for a specific hour, minute, and second
TimeValue	Returns a time
Weekday	Returns a number that represents the day of the week (between 1 and 7, inclusive)
WeekdayName	Returns the weekday name of a specified day of the week
Year	Returns a number that represents the year

Conversion Functions

Top

Function	Description
Asc	Converts the first letter in a string to ANSI code
CBool	Converts an expression to a variant of subtype Boolean
CByte	Converts an expression to a variant of subtype Byte
CCur	Converts an expression to a variant of subtype Currency
CDate	Converts a valid date and time expression to the variant of subtype Date
CDbl	Converts an expression to a variant of subtype Double
Chr	Converts the specified ANSI code to a character
CInt	Converts an expression to a variant of subtype Integer
CLng	Converts an expression to a variant of subtype Long
CSng	Converts an expression to a variant of subtype Single
CStr	Converts an expression to a variant of subtype String
Hex	Returns the hexadecimal value of a specified number
Oct	Returns the octal value of a specified number

Format Functions

Top

Function	Description
FormatCurrency	Returns an expression formatted as a currency value
FormatDateTime	Returns an expression formatted as a date or time
FormatNumber	Returns an expression formatted as a number
FormatPercent	Returns an expression formatted as a percentage

Math Functions

Top

Function	Description
Abs	Returns the absolute value of a specified number

Atn	Returns the arctangent of a specified number
Cos	Returns the cosine of a specified number (angle)
Exp	Returns *e* raised to a power
Hex	Returns the hexadecimal value of a specified number
Int	Returns the integer part of a specified number
Fix	Returns the integer part of a specified number
Log	Returns the natural logarithm of a specified number
Oct	Returns the octal value of a specified number
Rnd	Returns a random number less than 1 but greater or equal to 0
Sgn	Returns an integer that indicates the sign of a specified number
Sin	Returns the sine of a specified number (angle)
Sqr	Returns the square root of a specified number
Tan	Returns the tangent of a specified number (angle)

Array Functions

Top

Function	Description
Array	Returns a variant containing an array
Filter	Returns a zero-based array that contains a subset of a string array based on a filter criteria
IsArray	Returns a Boolean value that indicates whether a specified variable is an array
Join	Returns a string that consists of a number of substrings in an array
LBound	Returns the smallest subscript for the indicated dimension of an array
Split	Returns a zero-based, one-dimensional array that contains a specified number of substrings
UBound	Returns the largest subscript for the indicated dimension of an array

String Functions

Top

Function	Description
InStr	Returns the position of the first occurrence of one string within another. The search begins at the first character of the string
InStrRev	Returns the position of the first occurrence of one string within another. The search begins at the last character of the string
LCase	Converts a specified string to lowercase
Left	Returns a specified number of characters from the left side of a string
Len	Returns the number of characters in a string
LTrim	Removes spaces on the left side of a string
RTrim	Removes spaces on the right side of a string
Trim	Removes spaces on both the left and the right side of a string
Mid	Returns a specified number of characters from a string
Replace	Replaces a specified part of a string with another string a specified number of times
Right	Returns a specified number of characters from the right side of a string
Space	Returns a string that consists of a specified number of spaces
StrComp	Compares two strings and returns a value that represents the result of the comparison
String	Returns a string that contains a repeating character of a specified length
StrReverse	Reverses a string
UCase	Converts a specified string to uppercase

Other Functions

Top

Function	Description
CreateObject	Creates an object of a specified type
Eval	Evaluates an expression and returns the result
GetLocale	Returns the current locale ID
GetObject	Returns a reference to an automation object from a file
GetRef	Allows you to connect a VBScript procedure to a DHTML event on your pages
InputBox	Displays a dialog box, where the user can write some input and/or click on a button, and returns the contents
IsEmpty	Returns a Boolean value that indicates whether a specified variable has been initialized or not
IsNull	Returns a Boolean value that indicates whether a specified expression contains no valid data (Null)
IsNumeric	Returns a Boolean value that indicates whether a specified expression can be evaluated as a number

IsObject	Returns a Boolean value that indicates whether the specified expression is an automation object
LoadPicture	Returns a picture object. Available only on 32-bit platforms
MsgBox	Displays a message box, waits for the user to click a button, and returns a value that indicates which button the user clicked
RGB	Returns a number that represents an RGB color value
Round	Rounds a number
ScriptEngine	Returns the scripting language in use
ScriptEngineBuildVersion	Returns the build version number of the scripting engine in use
ScriptEngineMajorVersion	Returns the major version number of the scripting engine in use
ScriptEngineMinorVersion	Returns the minor version number of the scripting engine in use
SetLocale	Sets the locale ID and returns the previous locale ID
TypeName	Returns the subtype of a specified variable
VarType	Returns a value that indicates the subtype of a specified variable

VBScript Keywords

Keyword	Description
empty	Used to indicate an uninitialized variable value. A variable value is uninitialized when it is first created and no value is assigned to it, or when a variable value is explicitly set to empty. Example: dim x 'the variable x is uninitialized! x="ff" 'the variable x is NOT uninitialized anymore x=empty 'the variable x is uninitialized! **Note:** This is not the same as Null!!
isEmpty	Used to test if a variable is uninitialized. Example: If (isEmpty(x)) 'is x uninitialized?
nothing	Used to indicate an uninitialized object value, or to disassociate an object variable from an object to release system resources. Example: set myObject=nothing
is nothing	Used to test if a value is an initialized object. Example: If (myObject Is Nothing) 'is it unset? **Note:** If you compare a value to Nothing, you will not get the right result! Example: If (myObject = Nothing) 'always false!
null	Used to indicate that a variable contains no valid data. One way to think of Null is that someone has explicitly set the value to "invalid", unlike Empty where the value is "not set". **Note:** This is not the same as Empty or Nothing!! Example: x=Null 'x contains no valid data
isNull	Used to test if a value contains invalid data. Example: if (isNull(x)) 'is x invalid?
true	Used to indicate a Boolean condition that is correct (true has a value of -1)
false	Used to indicate a Boolean condition that is not correct (false has a value of 0)

VBScript Procedures

We have two kinds of procedures: The Sub procedure and the Function procedure.

A Sub procedure:

- is a series of statements, enclosed by the Sub and End Sub statements
- can perform actions, but **does not return** a value
- can take arguments that are passed to it by a calling procedure
- without arguments, must include an empty set of parentheses ()

```
Sub mysub()
 some statements
End Sub

or

Sub mysub(argument1,argument2)
 some statements
End Sub
```

A Function procedure:

- is a series of statements, enclosed by the Function and End Function statements
- can perform actions and **can return** a value
- can take arguments that are passed to it by a calling procedure
- without arguments, must include an empty set of parentheses ()
- returns a value by assigning a value to its name

```
Function myfunction()
 some statements
 myfunction=some value
End Function

or

Function myfunction(argument1,argument2)
 some statements
 myfunction=some value
End Function
```

Call a Sub or Function Procedure

When you call a Function in your code, you do like this:

```
name = findname()
```

Here you call a Function called "findname", the Function returns a value that will be stored in the variable "name".

Or, you can do like this:

```
msgbox "Your name is " & findname()
```

Here you also call a Function called "findname", the Function returns a value that will be displayed in the message box.

When you call a Sub procedure you can use the Call statement, like this:

```
Call MyProc(argument)
```

Or, you can omit the Call statement, like this:

```
MyProc argument
```

What is a Variable?

A variable is a "container" for information you want to store. A variable's value can change during the script. You can refer to a variable by name to see its value or to change its value. In VBScript, all variables are of type *variant*, that can store different types of data.

Rules for Variable Names:

- Must begin with a letter
- Cannot contain a period (.)
- Cannot exceed 255 characters

Declaring Variables

You can declare variables with the Dim, Public or the Private statement. Like this:

```
dim name
name=some value
```

Now you have created a variable. The name of the variable is "name".

You can also declare variables by using its name in your script. Like this:

```
name=some value
```

Now you have also created a variable. The name of the variable is "name".

However, the last method is not a good practice, because you can misspell the variable name later in your script, and that can cause strange results when your script is running. This is because when you misspell for example the "name" variable to "nime" the script will automatically create a new variable called "nime". To prevent your script from doing this you can use the Option Explicit statement. When you use this statement you will have to declare all your variables with the dim, public or private statement. Put the Option Explicit statement on the top of your script. Like this:

```
option explicit
dim name
name=some value
```

Assigning Values to Variables

You assign a value to a variable like this:

```
name="Hege"
i=200
```

The variable name is on the left side of the expression and the value you want to assign to the variable is on the right. Now the variable "name" has the value "Hege".

Lifetime of Variables

How long a variable exists is its lifetime.

When you declare a variable within a procedure, the variable can only be accessed within that procedure. When the procedure exits, the variable is destroyed. These variables are called local variables. You can have local variables with the same name in different procedures, because each is recognized only by the procedure in which it is declared.

If you declare a variable outside a procedure, all the procedures on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

Array Variables

Sometimes you want to assign more than one value to a single variable. Then you can create a variable that can contain a series of values. This is called an array variable. The declaration of an array variable uses parentheses ( ) following the variable name. In the following example, an array containing 3 elements is declared:

```
dim names(2)
```

The number shown in the parentheses is 2. We start at zero so this array contains 3 elements. This is a fixed-size array. You assign data to each of the elements of the array like this:

```
names(0)="Tove"
names(1)="Jani"
names(2)="Stale"
```

Similarly, the data can be retrieved from any element using the index of the particular array element you want. Like this:

```
mother=names(0)
```

You can have up to 60 dimensions in an array. Multiple dimensions are declared by separating the numbers in the parentheses with commas. Here we have a two-dimensional array consisting of 5 rows and 7 columns:

```
dim table(4, 6)
```

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In VBScript we have three conditional statements:

- **if statement** - use this statement if you want to execute a set of code when a condition is true
- **if...then...else statement** - use this statement if you want to select one of two sets of lines to execute
- **if...then...elseif statement** - use this statement if you want to select one of many sets of lines to execute
- **select case statement** - use this statement if you want to select one of many sets of lines to execute

---

If....Then.....Else

You should use the If...Then...Else statement if you want to

- execute some code if a condition is true
- select one of two blocks of code to execute

If you want to execute only **one** statement when a condition is true, you can write the code on one line:

```
if i=10 Then msgbox "Hello"
```

There is no ..else.. in this syntax. You just tell the code to perform **one action** if the condition is true (in this case if i=10).

If you want to execute **more than one** statement when a condition is true, you must put each statement on separate lines and end the statement with the keyword "End If":

```
if i=10 Then
   msgbox "Hello"
   i = i+1
end If
```

There is no ..else.. in this syntax either. You just tell the code to perform **multiple actions** if the condition is true.

If you want to execute a statement if a condition is true and execute another statement if the condition is not true, you must add the "Else" keyword:

```
if i=10 then
   msgbox "Hello"
else
   msgbox "Goodbye"
end If
```

The first block of code will be executed if the condition is true, and the other block will be executed otherwise (if i is not equal to 10).

If....Then.....Elseif

You can use the if...then...elseif statement if you want to select one of many blocks of code to execute:

```
if payment="Cash" then
   msgbox "You are going to pay cash!"
 elseif payment="Visa" then
   msgbox "You are going to pay with visa."
 elseif payment="AmEx" then
   msgbox "You are going to pay with American Express."
 else
   msgbox "Unknown method of payment."
end If
```

Select Case

You can also use the SELECT statement if you want to select one of many blocks of code to execute:

```
select case payment
 case "Cash"
   msgbox "You are going to pay cash"
 case "Visa"
   msgbox "You are going to pay with visa"
 case "AmEx"
   msgbox "You are going to pay with American Express"
 case Else
   msgbox "Unknown method of payment"
end select
```

This is how it works: First we have a single expression (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each Case in the structure. If there is a match, the block of code associated with that Case is executed.

Looping Statements

Very often when you write code, you want to allow the same block of code to run a number of times. You can use looping statements in your code to do this.

In VBScript we have four looping statements:

- **For...Next statement** - runs statements a specified number of times.
- **For Each...Next statement** - runs statements for each item in a collection or each element of an array
- **Do...Loop statement** - loops while or until a condition is true
- **While...Wend statement** - Do not use it - use the Do...Loop statement instead

---

For...Next

You can use a **For...Next** statement to run a block of code, when you know how many repetitions you want.

You can use a counter variable that increases or decreases with each repetition of the loop, like this:

```
For i=1 to 10
  some code
Next
```

The **For** statement specifies the counter variable (**i**) and its start and end values. The **Next** statement increases the counter variable (**i**) by one.

Step Keyword

Using the **Step** keyword, you can increase or decrease the counter variable by the value you specify.

In the example below, the counter variable (**i**) is increased by two each time the loop repeats.

```
For i=2 To 10 Step 2
```

```
  some code
Next
```

To decrease the counter variable, you must use a negative **Step** value. You must specify an end value that is less than the start value.

In the example below, the counter variable (**i**) is decreased by two each time the loop repeats.

```
For i=10 To 2 Step -2
  some code
Next
```

Exit a For...Next

You can exit a For...Next statement with the Exit For keyword.

For Each...Next

A **For Each...Next** loop repeats a block of code for each item in a collection, or for each element of an array.

The **For Each...Next** statement looks almost identical to the For...Next statement. The difference is that you do not have to specify the number of items you want to loop through.

```
dim names(2)
names(0)="Tove"
names(1)="Jani"
names(2)="Hege"

For Each x in names
  document.write(x & "<br />")
Next
```

Do...Loop

You can use Do...Loop statements to run a block of code when you do not know how many repetitions you want. The block of code is repeated while a condition is true or until a condition becomes true.

Repeating Code While a Condition is True

You use the While keyword to check a condition in a Do...Loop statement.

```
Do While i>10
  some code
Loop
```

If **i** equals 9, the code inside the loop above will never be executed.

```
Do
  some code
Loop While i>10
```

The code inside this loop will be executed at least one time, even if **i** is less than 10.

Repeating Code Until a Condition Becomes True

You use the Until keyword to check a condition in a Do...Loop statement.

```
Do Until i=10
  some code
Loop
```

If **i** equals 10, the code inside the loop will never be executed.

```
Do
```

```
  some code
Loop Until i=10
```

The code inside this loop will be executed at least one time, even if **i** is equal to 10.

Exit a Do...Loop

You can exit a Do...Loop statement with the Exit Do keyword.

```
Do Until i=10
  i=i-1
  If i<10 Then Exit Do
Loop
```

The code inside this loop will be executed as long as **i** is different from 10, and as long as **i** is greater than 10.