

Explanation of VBScript for use in HP QuickTest Professional (QTP)

Index

Sec.	Contents	Page
1	VBScript in QTP	2
2	VBScript Data Type	2
3	VBScript Variable	3
4	VBScript Arrays	4
5	VBScript Constants	4
6	VBScript Functions & Subroutines	7
7	Arguments in Procedures	10
8	VBScript Conditional Statements	11
9	VBScript Looping Statements	13
10	VBScript Classes	17
11	VBScript - Property Let, Property Get, Property Set	20
12	Example of VBScript - Property Let, Property Get, Property Set	23

Explanation of VBScript for use in HP QuickTest Professional (QTP)

1) VBScript in QTP

Scripting language for QuickTest Professional (QTP) is VBScript.

VBScript (short form of Visual Basic Scripting Edition) is a lively scripting language interpreted via Microsoft's Windows Script Host.

VBScript has many powerful functions and provides excellent support for variables, data types, and error handling.

Two script engines can interpret VBScript- **VBScript.dll**, which is invoked by **asp.dll** is used in web environment and **Wscript.exe & Cscript.exe** in Windows GUI environment using Windows Script Host (WSH).

We typically, use VBScript within WSH to automate systems administration tasks. WSH is the system module that transforms a VBScript file into a Windows executable file.

Wscript.exe is used to display output and receive input in **Windows** GUI format such as dialog and input boxes.

Cscript.exe is used in a command-line environment. When VBScript source code is contained in standalone files, they have the file extension **.Vbs**.

2) VBScript Data Type

VBScript has only one data type called a Variant that can store any type of value. A Variant is a special kind of data type that can contain different kinds of information, depending on how it is used.

Suppose if you're using a variable for assigning a numeric value, that variable behaves like a numeric data type. If you assign string value, that variable behaves like a string.

VBScript contains following subdatatypes in the Variant.

- 1) Empty
- 2) Null
- 3) Boolean
- 4) Byte
- 5) Integer
- 6) Currency
- 7) Long
- 8) Single
- 9) Double
- 10) Date (Time)
- 11) String
- 12) Object
- 13) Error

We can use conversion functions to convert data from one sub-data type to another type. To find subdatatype of a variant we need to use vartype function.

Explanation of VBScript for use in HP QuickTest Professional (QTP)

3) VBScript Variable

A variable is a convenient placeholder to store program information. We can change variable value in script running time. In VBScript variables are always of one fundamental data type Variant.

Advantage of using Variable in scripts: Variable is quite useful in carrying a value. For example if the script is using a value 10 in five places (3rd, 7th, 12th, 17th, 20th lines). In case the value is changed from 10 to 20 then we need to change that value in all the places wherever it is used. But if we have used variable in place of value (x=10) we need to change in only one place if that value is changed from 10 to 20(x=20). Variables are having flexibility to change value in run time.

Variable names go by the following standard naming conventions in VBScript. A variable name:

- # Must begin with an alphabetic character
- # Cannot contain an embedded period (.)
- # Must not exceed 255 characters
- # Must be unique in the scope in which it is declared.

It must be distinctive (unique) within the scope in which it is declared. If we declare a variable inside a procedure then its scope is local to that Procedure and only code within that procedure can access it, but if we declare a variable outside a procedure in a script, it is available to all the procedures in a script.

Procedure level variable exists as long as we are in the procedure and a life of a script level variable is the time it is declared up till the time script finishes.

Variables can be declared **explicitly** and **implicitly**.

Explicitly variables are declared with Dim statement, Public Statement, Private Statement.

```
Dim Name  
Dim Name, employee address, city
```

Implicitly we can declare them within the script by just using the variable name. But this practice is prone to errors.

We can force VBScript to require all variables to be explicitly declared by including the statement **Option Explicit** at the start of every script.

VBScript does not provide support for **constants**.

From VBScript 5.0 onwards programmers are able to create class constructs in VBScript-A step towards OOP.

Variables declared by Dim and Public are public in nature (accessible outside of the class). By default also variables are Public: in nature. With Private we can declare variables not visible outside of the class.

Example of a variable:

Type this in notepad, save the notepad with some name and .html extension (like in c:\program\p.html where "program" is the name of a folder)

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
<html>
<body>
<script type="text/vbscript">
dim variable_name
variable name ="Testing Genius"
document.write(variable_name)
</script>
</body>
</html>
```

Now open internet explorer and in the address bar type c:\program\p.html and press enter.

Another example that can be tried is:

```
<html>
<body>
<script type="text/vbscript">
dim variable_name
variable_name = 2
variable_name = variable_name+1
document.write(variable_name)
</script>
</body>
</html>
```

Another example of getting input from the user:

Enter the below code in notepad and save it with .vbs extension (like we saved it as c:\program\q.vbs where "program" is the name of a folder)

```
dim variable_name

variable name =InputBox("Enter your name:")

MsgBox("Your name is " & variable_name)
```

Now go to command prompt (C:\>) and type program\q and hit enter (there is no need to type the extension)

4) VBScript Arrays

Array Variable: A variable containing a single value is a scalar variable. we can create a variable that can contain a series of values using an index number. This is called an array variable. Arrays are useful when we are storing sets of similar data. We can store any kind of data in an array. The array can hold a combination of data types.

Arrays are of two types: 1) Fixed Length Arrays 2) Dynamic Arrays

Fixed arrays have a specific number of elements in them, whereas dynamic arrays can vary in the number of elements depending on how many are stored in the array.

Here we have explained all the ways to initialize and use arrays in VBScript. Every element of an array is associated with a unique index number. By default, index number starts from 0. The number of elements in an array is a fixed number. It can also be re-adjusted dynamically.

Explanation of VBScript for use in HP QuickTest Professional (QTP)

Method of creating Fixed Length Arrays:

For Example Dim a(10) : Here 'a' is an array and is having 11 elements (Since array count starts from 0). Here it's a fixed size array with size 10.

Method of creating Dynamic Arrays:

A dynamic array is created in the same way as a fixed array, but we don't put any bounds in the declaration.

For Example Dim x() : Here 'x' is the dynamic array and we can store n number of elements in it. The benefit of a dynamic array is that if we don't know how large the array will be when we write the code, we can create code that sets or changes the size while the VBScript code is running.

Arrays can have multiple dimensions. VBScript supports up to **60**.

Example – 1:

```
Dim variable_name(upper_limit) [As data_type]
```

If "As data_type" is not specified, it will be a variant. Above we have declared a fixed size array. The array size limit is upper limit +1 because index starts from 0.

```
<html>
<body>
<script type="text/vbscript">
dim e_name (5)
e_name(0)="Himanshu"
e_name(1)="Piyush"
e_name(2)="Neha"
e_name(3)="Manjri"
e_name(4)="Rohit"
e_name(5)="Amar"
for i=0 to 5
document.write(e_name(i) & "<br />")
next
</script>
</body>
</html>
```

Example – 2:

```
Dim variable_name() [As data-type]
ReDim [Preserve] variable_name(upper_limit)
```

Firstly we declare an array with no upper limit and then with **ReDim** we reset the upper bound to a new value. The optional key word "**Preserve**" states that all of the old elements must be preserved when changing the array size.

The size of the dynamic array changes during the time our script is running. The array is initially declared using either the Dim statement or using the **ReDim** statement, For a dynamic array, no size or number of dimensions is placed inside parentheses.

```
Dim first Array()
ReDim second-Array()
```

In the following example, ReDim sets the initial size of the dynamic array to 25

```
ReDim first_Array(25)
```

We can resize a dynamic array unlimited number of times.

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
Dim array-dynamic()
```

```
' Size the dimension to contain one dimension with 3 elements
```

```
ReDim array_dynamic(2)
```

```
Put data in the array
```

```
array_dynamic(0)= "1"  
array_dynamic(1)= "2"  
array_dynamic(2)= "3"
```

```
Resize the array, but keep the existing data
```

```
ReDim Preserve array_dynamic(5)
```

```
' Display the 3rd element
```

```
MsgBox array_dynamic(2)  
MsgBox displays 3.
```

Example – 3:

```
variable_name = Array(element1, element2, ...)
```

Array function takes values of variant type and returns a dynamic sized array. The arguments are a listing of values that will become the elements of the array,

```
Dim a  
a=Array(5,10,15,20)  
document.write(a(3))
```

Output: 20

Some of the Array keywords and their uses;

Keyword	Function
Dim	It will Declare an array
Erase	Reinitializes the elements if it is a fixed-size array and deallocates the memory used if it is a dynamic array.
IsArray	This will Return True if A is an array, False if it is not
LBound	This will Return lower bound of an array, in VBScript it will always return 0
Preserve	Preserve (Optional) is used to preserve the data in an existing array, when you resize it.
ReDim	This is used to size or resize a dynamic array.
UBound	This will Return all upper bound of array

Explanation of VBScript for use in HP QuickTest Professional (QTP)

5) VBScript Constants

A constant is a meaningful name that takes the place of a number or string and never changes. The difference between variable and constant is we can change the variable value in run time but for constants its not possible.

How do we create constants?

For Example: `const str="QTPGENIUS"` : Here "str" is a constant and the value will never change.

We have two types of constants 1) Public constants 2) Private constants.

By default all constants are Public. However we can specify the type if required

For Example: `Public const str="QTPGENIUS"`

or

For Example: `Private const str="QTPGENIUS"`

6) VBScript Functions and Subroutines

There are two types of procedures in VBScript

1) Function Procedure: It is a series of VBScript statements enclosed by the Function and End Function statements. In Function procedures we can use function name to assign a value. Function Procedure is able to return the value.

For example:

```
Function demo_add(a,b)
demo_add=a+b
End Function
oVal=demo_add(2,3)
msgbox oVal 'Returns 5
```

In this example `demo_add` function returns a value to `oVal`.

2) Sub Procedure: It is a series of VBScript statements enclosed by the Sub and End Sub statements. Sub Procedure cannot return any value.

For example:

```
Sub demo_sub(a,b,c)
c=a+b
End sub
demo_sub 2,3,x
msgbox x 'Returns 5
```

This example will do the same as what function procedure is doing above. But in sub Procedure we need to use one more parameter to get values from the sub procedure.

The main difference between a function and a subroutine is that a subroutine will do some processing of the code and then quit; while a function processes some code and then returns the result back.

Explanation of VBScript for use in HP QuickTest Professional (QTP)

VBScript functions are described using the Function and End Function keywords.

```
<html>
<body>
<script type="text/vbscript">
FUNCTION add2numbers
    Result . 2 + 2
    add2numbers = Result
END FUNCTION
document.write(add2numbers)
</script>
</body> </html>
```

Adding numbers by passing parameters:

```
<html>
<body>
<script type="text/vbscript">
FUNCTION ADD(Number1, Number2)
    Result = Number1 + Number2
    ADD = Result
END FUNCTION
document.write(ADD(4, 1))
</script>
</body>
</html>
```

Function to get square of a number:

```
<html>
<body>
<script type="text/vbscript">
Function do_square(number)
    Do_square = number + number
end function
'first way of calling a function
document.write(do_square(5))
document.write(" ")
'second way of calling a function
if 90 < do_square(8) then
    document.Write("90 is less than 8*8")
else
    document.Write("8*8 is less than 90")
end if
</script>
</body>
</html>
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

A Sub procedure is a series of VBScript statements, enclosed by Sub and End Sub statements

```
<html>
<body>
<script type="text/vbscript">
call square ()
Sub square()
    Var1 = InputBox("Please enter a number", 1)
    MsgBox ("The square is " & do_square(var1))
End Sub
Function do_square(number)
    do_square = number * number
end function
</script>
</body>
</html>
```

We have written the calling statement (call square()) in the script itself.

Passing variable by reference example:

```
<html>
<head>
<script type="text/vbscript">
sub mysub ( )
dim x
x=2
document.write("in mysub" & x )
call newsub(x)
document.write("Back in mysub" & x )
end sub
call mysub()
sub newSub(byref var1)
var1=var1*var1
document.write("in newsub" & var1)
end sub
</script>
</head>
</html>
```

This shows that only the address is being passed that is why it is showing the updated value third time in Back in mysub.

Passing variable by value example:

```
<html>
<head>
<script type="text/vbscript">
sub mysub ( )
dim x
x=2
document.write("in mysub" & x )
call newsub(x)
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
document.write("Back in mysub" & x )  
end sub  
call mysub()  
sub newSub(byval var1)  
var1=var1*var1  
document.write("in newsub" & var1)  
end sub  
</script>  
</head>  
</html>
```

This shows that another value is being passed as can be seen by the result of the third x as in Back in mysub.

7) Arguments in Procedures

There are two types of arguments in procedures

- 1) **By Val:** Indicates that the argument is passed by value.
- 2) **By Ref:** Indicates that the argument is passed by reference.

By default all arguments are 'ByRef'.

For example:

```
Function demo_add(a,b)  
demo_add=a+b  
End Function
```

Here 'a,b' are the arguments. By default these are 'ByRef'.

In simple terms 'ByRef' Means the value which is assigned to the variable with in the function is permanent and we can use that value out side that function as well.

'ByVal' means the value, which is assigned to the variable with in the function, is temporary and we can use that value only with in that function.

For example:

```
Function demo_parameters(byref x,byval y)  
x=20  
y=50  
demo_parameters=x+y  
End Function
```

```
a=10  
b=20  
msgbox demo_parameters(a,b)  
msgbox a  
msgbox b
```

In the above function 'x' and 'y' are the arguments, declared as 'byref' and 'byval'. With in that function we have assigned values to 'x' and 'y'.

Outside of the function we have assigned values to two variables and passing those variables in to the

Explanation of VBScript for use in HP QuickTest Professional (QTP)

function. Here 'a' is passing reference to x and b is passing value to y. With in that function we are changing the value for 'x'. This value is permanent for 'a' - because 'a' is passed as 'ByRef'. But the value of 'b' will not be changed because it is passed as 'ByVal'.

8) VBScript Conditional Statements

Three types of conditional statements are there in VBScript.

if ... then ... else statement

Example of if statement without else:

Example – 1:

```
if i=2 Then msgbox "Hello World"
```

Example – 2:

```
<script type="text/vbscript">
Dim var_num
var_num = 7
If var_num = 7 Then
    document.write("You are a Lucky 7!")
End If
var_num = 2
If var_num = 7 Then
    docuxent.write("Hello!")
End If
</script>
```

Example – 3:

More than one statement can be executed for truth condition by putting the statements on separate line.

```
if i=2 Then
    msgbox "Hello World"
    i = i+1
end if
```

Example of ifthen.... else statement:

Example – 1:

```
if i=2 Then
    msgbox "Hello World"
else
    msgbox "Thank You"
```

Example – 2:

```
<script type="text/vbscript">
Dim weight
weight = 50
If weight > 70 Then
    document.write("You are Over Weight")
Else
    document.write("Your Weight is perfect")
End If
</script>
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

Example of ifthen.... elseif statement:

Example – 1:

```
if fee="Cash" then
    msgbox "pay cash!"
elseif fee="Visa" then
    msgbox "pay with visa."
elseif fee="American Express" then
    msgbox "pay with American Express."
else
    msgbox "Unknown method of payment"
End If
</script>
```

Example – 2:

```
<script type="text/vbscript">
Dim weight
weight = 65
If weight > 70 Then
    document.write("You are Over Weight")
elseif weight > 60 Then
    document.write("Your Weight is Perfect")
elseif weight > 50 Then
    document.write("You are under Weight")
Else
    document.write("Do Weight again")
End If
</script>
```

Select case statement

Example – 1:

```
select case fee
case "Cash"
msgbox "pay cash"

case "Visa"
msgbox "pay with visa"

case "American Express"
msgbox "pay with American Express"

case Else
msgbox "Unknown method of payment"

end select
```

Example – 2:

```
<script type="text/vbscript">
Dim myName
myName = "Rajesh"
Select case myName
Case "Manoj"
    document.write("Are you there Manoj?")
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
Case "Pooja"  
    document.write("Hello Pooja?")  
Case " Rajesh"  
    document.write("How are you Rajesh ?")  
End Select  
</script>
```

Example – 3:

```
<script type="text/vbscript">  
Dim myName  
myName = "Rajesh"  
Select case myName  
Case "Manoj"  
    document.write("How are you Manoj?")  
Case "Pooja"  
    document.write("Where are you Pooja?")  
Case " Rajesh"  
    document.write("Where did you go Rajesh ?")  
Case Else  
    document.write("Who are you?")  
End Select  
</script>
```

A single expression (usually variable) is evaluated once and its value is then compared with the values for each case. If there is a match, the block of code associated with that case is executed. If there is no match then Else case is executed.

9) VBScript Looping Statements

We have four looping statements in VBScript

For....Next Statement

Example – 1:

```
<script type="text/vbscript">  
For num = 0 to 3  
    document.write("<br />Loop #" & num)  
Next  
</script>
```

```
Loop #0  
Loop #1  
Loop #2  
Loop #3
```

With the help of Step keyword, we can increase or decrease the counter by the value specified.

Example – 1:

```
For i=2 To 8 Step 2  
    any piece of code  
Next
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

Example – 2:

```
For i=8 To 2 Step 2
    any piece of code
Next
```

For Each ... Next statement

It is useful when we don't know how many elements are there in the array.

Example – 1:

```
<script type="text/vbscript">
Dim cupboard (2)
cupboard(0) = "Trousse"
cupboard(1) = "Shirt"
cupboard(2) = "Suit"
    document.write("In my cupboard is:")
For Each y In cupboard
    doclment.write(y & "<br>")
Next
</scripts
```

Example – 2:

```
dim names(2)
names(0)="Rajan"
names(1)="Suresh"
names(2)="Reena"
For Each x in names
    document.write(x & "")
Next
```

Do...Loop

It will repeat a block of code while a condition is True or until a condition becomes True

Example – 1:

```
Do While i>9
    any piece of code
Loop
```

If i equals 8, the code inside the loop above will never be executed.

Example – 2:

```
Do
    any piece of code
Loop While i>9
```

The code inside this loop will be executed at least one time, even if i is less than 9.

Repeating Code Until a Condition Becomes True

Explanation of VBScript for use in HP QuickTest Professional (QTP)

Example – 1:

```
Do Until i=9
    any piece of code
Loop
```

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

Example – 2:

```
Do
    any piece of code
Loop Until i=9
```

The code inside this loop will be executed at least one time, even if i is equal to 9.

The Exit statement can only be used within a Do ... Loop control structure to provide an alternate way to exit a Do ... Loop.

We must end all Do statements with Loop or otherwise error message will Pop up. The While and the Until condition may be placed after the Do or the Loop.

Some Examples:

Example – 1:

```
num = 1
Do
    num = num + 1
Loop Until num = 5
```

Example – 2:

```
num = 1
Do While num < 5
    num = num + 1
Loop
```

Example – 3:

```
num = 1
Do
    num = num + 1 br>Loop While num < 5
Exit a Do ... Loop
```

We can exit a **Do ... Loop** statement with the Exit Do keyword.

Example – 1:

```
Do Until i=9
    i=i-1
If i<9 Then Exit Do
Loop
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

The code inside this loop will be executed as long as i is different from 9, and as long as i is greater than 9.

While-Wend statement

While Loop is a simple loop that keeps looping while a condition is true

Example – 1:

```
<script type="text/vbscript">
Dim count
count = 9
While count > 0
    document.write (count)
    document.write ("<br />")
    count = count - 1
Wend
    document.write("Finish!")
</script>
```

```
9
8
7
6
5
4
3
2
1
Finish!
```

More VBScript techniques can be accessed by visiting the following link “10 VBScript techniques”
<http://windowsitpro.com/article/articleid/20979/10-more-vbscript-techniques.html>

Michael Otey the author of “10 More VBScript Techniques” states that

“In my June 2001 column, I shared 10 basic VBScript techniques. For those who want to step up a level and begin writing productive administrative scripts, here are 10 more VBScript techniques.

10. On Error-The On Error statement lets a script trap runtime errors and continue executing. You can test for errors in the script after each statement has executed. On Error Resume Next

9. InStr -This function lets you locate a substring in a string. The function returns the starting position of the substring or a 0 if the function doesn't find the string. In

```
nPos = InStr("123345", "33")
```

nPos has a value of 3 because "33" begins in the third position of "123345."

8. The Do Loop -This basic mechanism for repeatedly executing a set of statements comes in two forms: a Do Until Loop and a Do While Loop. The most important distinction between the two loops is that the Do Until Loop arrays executes at least once.

```
Do Until myValue > 1
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
myValue = myValue + 1
Loop
```

7. Subroutines – Modularizing your code into subroutines lets you organize your scripts and create reusable routines. You can define subroutines anywhere in a script. You use subroutines when you don't need to return a value to the calling code. . . .

10) VBScript Classes

```
Class Hello.World
Public Sub Say_Hello(Name)
MsgBox "Hello, " & Name & ", welcome to " & Garden &".
End Sub
Public Garden
End Class
```

```
Dim MyHello_World
Set MyHello_World = New Hello_World
MyHello_World.Garden = "Fountain"
MyHello_World.Say_Hello "Genius"
```

Above we have created a class (Hello_World) and an instance (MyHello_World) of that class. VBScript uses the Class...End Class statements to define the contents of the class. The property (Garden) and procedure (Say Hello) are also declared within the class. Write the whole code written above in notepad, save it as .vbs and run it.

Members within the class can be declared as private and Public. Private members are only visible within the class whereas public members are accessible by any code outside of the class. Public is default.

Procedures (Sub or Function) declared Public within the class are methods of the class. Public variables serve as properties of the class,

Property Let will allow code outside of the class to assign a value to a private variable of the class.

```
Class A
Private name
Public Property Let assign_name(e_Name)
name = e_Name
End Property
End Class
```

A Property Let procedure must accept at least one argument. This procedure can also handle the process of data validation to check some validation e.g. if the value you are assigning is more than 5 characters long or not.

Property Get will allow code outside of the class to read the value of a Private property variable.

```
Class A
Private name
Public Property Let assign_name(e_Name)
name = e_Name
End Property

Public Property Get assign_name()
assign_name = name
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
End Property  
End Class
```

The **Property Get** procedure does not accept any arguments, however VBScript allows you to add an argument. For this you have to add an additional argument to the property's corresponding Property Let or Property Set procedure because a Property Let/Set procedure must always have exactly one more argument than its corresponding Property Get procedure.

Property Set - This is an object property assignment procedure used to assign the New Property value to the private object variable (if the private variable is an object). Following **op_sys** is an object read-write property.

```
Class Machine  
Private obj_oOS  
Public Property Set op_sys(oObj)  
Set obj_oOS = oObj  
End Property  
Public Property Get op_sys( )  
Set op_sys = obj_oOS  
End Property  
End Class
```

We can make a property Read-Only by following two methods:

Method-1: By writing only a Property Get procedure for the property:

In the absence of a Property Let procedure, code outside of the class cannot write to the employeeName property.

```
Class employee  
Private ename  
Public Property Get employeeName()  
employeeName = ename  
End Property  
End Class
```

Method-2: By declaring the Property Get procedure as Public and the Property Let procedure as Private:

```
Class employee  
Private ename  
Private Property Let employeeName(strName)  
ename = strName  
End Property  
Public Property Get employeeName()  
employeeName = ename  
End Property  
End Class
```

Class Methods:

Where functions or procedures are written inside the class they are called methods.

If a class method is declared as Public there it will be available to code outside or inside the class, and a method that is declared as Private will be available only to code inside the class.

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
Class welcome
  Private ur_name
  Public Property Let Name(var_name)
    ur_name =var_name
  End Property
  Public Sub Showwelcome(var_type)
    MsgBox Makewelcome(var_type) & Ur_name & "."
  End Sub
  Private Function Makewelcome(var_type)
    Select Case var_type
    Case "Formal"
      Makewelcome = "welcome,"
    Case "Informal"
      Makewelcome = "Hello there, "
    Case "Casual"
      Makewelcome = "Hey, "
    End Select
  End Function
End Class
```

```
Dim my_object
Set my_object = New welcome
With my_object
  .Name = "Genius"
  .Showwelcome "Informal"
  .Showwelcome "Formal"
  .Showwelcome "Casual"
End With
Set my_object = Nothing
```

Class Events:

Class_Initialize and Class_Terminate are associated with every class that we create. Class_Initialize is used whenever an object based on a class is instantiated. e.g.

```
Set objectname = New classname
```

Class_Initialize event's general format is:

```
Private Sub Class_Initialize( )
  'Initialization code goes here
End Sub
```

The Class Terminate event is used when the object goes out of scope, or when the object is set to nothing.

Class Terminate event's general format is:

```
Private Sub Class_Terminate( )
  'Termination code goes here
End Sub
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

Another example of a class:

```
<html>
<head>
<script type="text/vbscript">
Class calculator
  Public Function add(var_first, var_second)
  Dim output
  output = var_first + var_second
  add = output
  End Function

  Public Function subtract(sub_first, sub_second)
  Dim output
  output = sub_first - sub_second
  subtract = output
  End Function
End Class
Dim calc_Object
Dim var_addition
Dim var_subtract
Set calc_object = New calculator
var_addition = calc Object_add(1,2)
var_subtract = calc Object.subtract(2,1)
document.write (var_addition)
document.write (var_subtract)
</script>
</body>
</html>
```

Below we will see file system objects of VBScript which allows us to access, copy, open, delete (and much more) files on the operating system.

[http://msdn2.microsoft.com/en-us/library/aa711216\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/aa711216(VS.71).aspx)

Summary of VBScript Class Concepts

1. It does NOT support inheritance. So it is not possible to create a Collie class which inherits characteristics from a Dog class which inherits characteristics from a Mammal class, etc.
2. It does NOT support polymorphism. Kind of a related to inheritance, where for example a Dog class will "bark" and a Pig class will "oink" when they are asked to "speak" (in a class hierarchy where they inherited the base Speak() method from the Mammal class, then override that method to implement animal unique behavior).
3. It DOES support encapsulation, an OO technique which conceals how a particular" class is implemented, independent from objects that use the class's Public Properties and Methods. Another way to think about encapsulation is to say it is the ability to hide implementation details while sharing higher level behavior.

In our opinion the lack of inheritance and polymorphism are not major shortcomings in scripting environments such as WSH and especially QTP, where we are not trying to build large complex OO programs.

Encapsulation is then the primary reason to consider using VBScript classes. And, with encapsulation comes namespace control which permits any number of class elements to be named for as long as each of those elements resides in a different class (i.e. a different namespace).

Explanation of VBScript for use in HP QuickTest Professional (QTP)

11) VBScript- Property Let, Property Get, Property Set

Class properties in VBScript are used to assign Values to private variable and handle the process of data validation.

Property Let: Which is used by the outside code to store a value in the private property variable. It is similar to a procedure in the sense that it does not return a value. A Property Let procedure Must accept at least one argument. If the private variable you are using is an object then the process of assignment and data validation is handled by Property bet.

Property Set: Similar to Property Let but used for object based properties. By default, the Property Set Procedure is Public.

To retrieve the value of a private variable we will retrieve the value of a Property.

Property Get: This is used by code outside of our class to read the value of a private property variable. It is similar to a function in the sense that it returns a value to the calling code -- this value is the private variable value.

The Property comet procedure does not accept any arguments. We can add an argument to it, but then we have to add an additional argument to the property's corresponding Property Let or Property Set procedure, because Property Let/Set procedure must always have exactly one more argument than its corresponding Property Get procedure.

If the property get procedure returns an object then we can use the set statement (but it works well without set also) to return the value.

```
Class ABC
  Private object
  Private var_obj
  Public Property Get username()
    Set username = var_obj
  End Property
End Class
```

Read only Properties have only Property Get procedure

Write-only properties have only a Property Let or a Property Set procedure

Read-Write properties have a Property Get procedure and either a Property Let or a Property Set procedure

Example - 1 of Property Let, Property Get, Property Set

Following Example, shows a simple class that defines a private variable, m_var, and a two read-write properties, one_type and two_type, the latter of which is an object property.

```
Class Computer
  Private m_var
  Private o_var

  Public Property Let one_type(stringtype)
    m_var = stringtype
  End Property
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
Public Property Get one_type( )
    one_type = m_var
End Property

Public Property Set two_type(cObj)
    Set o_var = oobj
End Property

Public Property Get two_type( )
    Set two_type = o_var
End Property

End Class
```

Example - 2 of Property Set

Here is the syntax for a Property Set procedure.

```
Class Main_class
    'Private FS_object object
    Private var_Obj
    Public Property Set FSPRO (objFSPRO)
        Set var_Obj = objFSPRO
    End Property
End Class
```

For example, here is what code that is using an object based on the above class might look like.

```
Dim objMain_class
Dim objFSPRO
Set objFSPRO
    WScript.CreateObject ("Scripting.FS_Object")
Set objMain_class = New Main_class
Set =objMain_class.FSPRO = objFSPRO
```

Last line uses the Set Statement when it writes to the FSPRO property. This is required because the Main_class class used a Property Set procedure for the FSPRO property. Without the Set statement at the beginning of the last line, VBScript would produce an error. When a property on a class is object based, it is typical to use a Property Set procedure. Most programmers using this class would expect this.

Example - 3 of Property Set

For example imagine we had a class that contained a private property named ob_var_conn that was expected to be an ADO Connection object. This class definition, with the property Set and Property Get Statements might look like:

```
Class Connect_Class
    'Create a private property to hold our Connection object
    Private ob_var_conn
    Public Property Get Connection ( )
        Set Connection = ob_var_conn
    End Property
    Public Property Set Connection (ob_var_ccnection)
        'Assign the private property ob_var_conn to ob_var_connection
        Set ob_var_conn= ob_var_connection
    End Property
End Class
```

Explanation of VBScript for use in HP QuickTest Professional (QTP)

```
End Property  
End Class
```

The end developer would use the Property Set statement in the following manner:

```
'Create an instance of Connect_ Class  
Dim ob_var_class, ob_var_record  
Set ob_var_class= New Connect_ Class  
Set ob_var_Connection = Server.CreateObject('ADODB.Connection')  
'Assign ob_var_Connection to the Connection property  
Set ob_var_class.Connection = ob_var_Connection
```

As with the Property Let statement, the Property Set statement has an optional argument list. This argument list must be identical to the corresponding Property Get's argument list.

12) Example of VBScript - Property Let, Property Get, Property Set

```
class PencilClass  
Private recentPencil, recentColor  
Property Get Pencil ()  
    Set Pencil = RecentPencil  
End Property  
Property Set Pencil (x)  
    Set recentPencil = x  
End Property  
Property Get Pencilcolor ()  
    Select Case recentColor  
        Case 1: Pencilcolor = "Orange"  
        Case 2: Pencilcolor = "Green"  
        Case 3: Pencilcolor = "Yellow"  
    End Select  
End Property  
Property Let Pencilcolor ()  
    If x = "Orange" Then  
        recentColor =1  
    Else  
        If x = "Green" Then  
            recentColor =2  
        Else  
            recentColor =0  
        EndIf  
    EndIf  
End Property  
End Class  
Set one_pencil = New PencilClass  
Set two_pencil = New PencilClass  
One_pencil.Pencilcolor = "Orange"  
wscript.echo One_pencil.Pencilcolor  
Set two_pencil.pencil = One_pencil 'Invokes Property Set  
wscript.echo "1st time" & one_pencil, pencilcolor  
two_pencil.pencil,Pencilcolor = "Green"  
wscript.echo "2nd time" & one_pencil, pencilcolor
```

References: VBScript compilation by experts on QTP like 1) Mr. Sachin Dhall and 2) Mr. Piyush Gupta