Sesame Script

Variables and Constants

By The Microsoft Scripting Guys

You know that old saying, "ask and ye shall receive"—you know, the one that rarely seems to work? (It certainly didn't work when the Scripting Guys asked for a six week vacation in Hawaii.) Well, this time it actually worked. You asked for more introductory articles on scripting, so we're presenting you with a new series of monthly articles aimed at beginning scripters. Welcome to *Sesame Script*!

On This Page

- What Are We Doing Here?
- Is There an Echo in Here?
- Variables
- What Do We Need Variables For?
- Hungarian Variables
- A Constant Reminder
- Advanced Constants
- VBScript Constants
- More Advanced Constants
- Scripting Puzzles

What Are We Doing Here?

We Scripting Guys ask ourselves this every day. What's worse is when our managers ask us "What are you doing here?" and we don't have an answer. But as to why this article is here, well, we already explained that… you voted, and we gave in to the overwhelming majority. Now, within those votes for more beginner material there was a lot of variation as to why people wanted more and what more they wanted. We heard from absolute beginners who know nothing about scripting, and from computer programmers who don't know the particulars of writing scripts, and everything in between. So how do we make everyone happy?

Well, we probably don't, but we're going to do our best. So keep sending us email and letting us know what you'd like to see here. Also, keep in mind the Script Center has a lot of other great featured articles to help you learn more about scripting. Scroll down to the Regular Features section on the Script Center Home page to find out more.

Today we're going to start with a brief explanation of the Wscript.Echo command that you saw in our introductory article Scripting: Your First Steps. From there we're going to start introducing you to some basic scripting concepts that will help you in writing and modifying scripts. We'll also throw in a couple of examples that are a little more complicated and tell you where you can read more about them. And don't miss our new bonus feature! (You have to read all the way to the end to find out about it. Okay, you don't actually have to read to the end, you could just scroll down, but you wouldn't learn much in the process. What fun would that be?)

Top of page

Is There an Echo in Here?

Let's start with a short explanation of a very important scripting command: **Wscript.Echo**. We saw this in our first article, but we didn't actually explain it at the time. It's pretty simple. Suppose you've written a script that gets the current date and time:

```
Now
```

Yes, that's it. That one little word will run as a VBScript and get the date and time. The only problem is that the computer has retrieved the system date and time, but hasn't bothered to tell you what it is. Kind of like when you tell your children they can't watch TV, then have to tell them to actually turn the TV off. You need to be very explicit with computers in the same way. Saying "get the date for me" doesn't also say "tell me what it is." That's what Wscript.Echo is for:

```
Wscript.Echo Now
```

Now you can actually see the current date and time. Wscript.Echo can display a lot of different types of information, anything from a simple error message to the results of an Active Directory query. Play around with it a little; try this script and see what you get back:

```
Wscript.Echo "Line One"
Wscript.Echo 100
Wscript.Echo 100 + 100
Wscript.Echo "100 + 100"
Wscript.Echo Now
Wscript.Echo "Now"
```

Note: Remember from our first article that Wscript.Echo will display its output to a message box if you run your script without using cscript. Running from the command line using cscript will send your output to the command window. There is a method named MsgBox that will always send output to a message box no matter how you run your script. We'll use this method a little later in this article.

More Information

Handling Input and Output – Microsoft Windows 2000 Scripting Guide

Echo Method –Windows Script Host Reference

Top of page

Variables

Now that we have that important step out of the way, we're going to take you back to a time you've long forgotten, or probably wish you could forget. For some of you we'll be going back a couple of hours, for others a couple of decades. We're going back to high school, to the dreaded algebra class. (Stay with us here, deep breath, everything's okay.)

Two trains are coming towards each other, one going 30 MPH,…no, no, we're not talking about anything like that (thank goodness!). We're starting with the very basics: 1 + 1 = 2.

Still with us? Good. Now, if we don't know what 1 + 1 equals and we want you to tell us, we'd say

```
1 + 1 = x
```

and ask you to tell us what x is. Scripting is pretty much the same way, only we turn things around a bit:

```
x = 1 + 1
```

Here we're just asking the computer to put the value of 1 + 1 into x. So if we ask the computer what x is, it will tell us 2. Try it in a script:

```
x = 1 + 1
Wscript.Echo x
```

In this script, x is called a variable. Which makes sense, because the value contained in x can vary:

```
x = 1 + 1
Wscript.Echo x
x = 2 + 2
Wscript.Echo x
```

Run this script and your output will be

```
2
4
```

As you can see, x is just a placeholder for whatever value we're asking the computer for.

Now you might be wondering, what's so special about x? Why not a or z? Well, why not?

```
a = 1 + 1
Wscript.Echo a
```

Run this script and you'll see it works the same with a as it did with x. Or how about this one:

```
TheValueOfOnePlusOne = 1 + 1
Wscript.Echo TheValueOfOnePlusOne
```

It doesn't really matter what you use for your variable. It helps to give the variable a name that will remind you what value it's holding, but other than that you can make up pretty much whatever name you want. (Okay, there are just a few restrictions: must begin with an alphabetic character, no periods, less than 256 characters, and can't be the same as a predefined VBScript keyword. If you want a variable name that's longer than 255 characters, well, you have more problems than we can help you with here.)

Top of page

What Do We Need Variables For?

You might be wondering why we even needed the variable. Why not just write a script like this:

```
Wscript.Echo 1 + 1
```

Yes, you could do that, and it would work just fine, giving you a value of 2. But a script that actually performs

useful tasks will typically be a little more complicated than this example, in which case variables come in pretty handy. One common use of variables is to put a value into the variable and then use that variable throughout the script. For example, you'll see many scripts on the Script Center with this line in them:

```
strComputer = "."
```

The dot represents the local computer. We've put this dot into the variable called strComputer. Here's a script that tells you what operating system is running on the local computer:

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:" _
 & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")

Set colOSes = objWMIService.ExecQuery("Select * from Win32_OperatingSystem")
For Each objOS in colOSes
  Wscript.Echo "Computer Name: " & objOS.CSName
  Wscript.Echo "Caption: " & objOS.Caption 'Name
  Wscript.Echo "Version: " & objOS.Version 'Version & build
  Wscript.Echo "Build Number: " & objOS.BuildNumber 'Build
  Wscript.Echo "Build Type: " & objOS.BuildType
  Wscript.Echo "OS Type: " & objOS.OSType
  Wscript.Echo "Other Type Description: " & objOS.OtherTypeDescription
Next
```

(Don't worry too much about the details here. As you can see it's mostly a bunch of Wscript.Echo statements anyway, which you already know all about.) We could just as easily have gone without the variable and just put a dot in wherever you see strComputer:

```
Set objWMIService = GetObject("winmgmts:" _
 & "{impersonationLevel=impersonate}!\\" & "." & "\root\cimv2")

Set colOSes = objWMIService.ExecQuery("Select * from Win32_OperatingSystem")
For Each objOS in colOSes
  Wscript.Echo "Computer Name: " & objOS.CSName
  Wscript.Echo "Caption: " & objOS.Caption 'Name
  Wscript.Echo "Version: " & objOS.Version 'Version & build
  Wscript.Echo "Build Number: " & objOS.BuildNumber 'Build
  Wscript.Echo "Build Type: " & objOS.BuildType
  Wscript.Echo "OS Type: " & objOS.OSType
  Wscript.Echo "Other Type Description: " & objOS.OtherTypeDescription
Next
```

But what if you want the script to run against a remote computer? You need to find the dot in the script, and replace it with the name of the remote computer. And you need to make sure you find every instance of that dot. In this script there's only one, but that won't always be the case. You can probably see the potential for errors in all this. By putting the computer name into a variable, all you have to do is switch out the computer names, and you don't have to worry about where it's used. So you could have a script that runs against the current computer, but by simply changing out the dot for a computer name (in this case server1), you can run the same script against a remote computer:

```
strComputer = "server1"
Set objWMIService = GetObject("winmgmts:" _
 & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")

Set colOSes = objWMIService.ExecQuery("Select * from Win32_OperatingSystem")
For Each objOS in colOSes
  Wscript.Echo "Computer Name: " & objOS.CSName
  Wscript.Echo "Caption: " & objOS.Caption 'Name
  Wscript.Echo "Version: " & objOS.Version 'Version & build
  Wscript.Echo "Build Number: " & objOS.BuildNumber 'Build
  Wscript.Echo "Build Type: " & objOS.BuildType
  Wscript.Echo "OS Type: " & objOS.OSType
```

```
  Wscript.Echo "Other Type Description: " & objOS.OtherTypeDescription
Next
```

The only thing that changed is the very first line.

And that's what variables are for.

More Information

Variables – Windows 2000 Scripting Guide

VBScript Variables – VBScript User's Guide

Top of page

Hungarian Variables

You might have heard about something called "Hungarian notation." Or maybe you've never heard of it, but you've probably seen it. It's when you have variables that look something like this:

```
intComputers = 20
dtToday = #07/01/05#
strComputerName = "server1"
```

Naming variables like this is a common programming practice. All you're doing is coming up with a variable name, such as intComputers, then prefixing it with the type of information you'll be storing in that variable. So if you're using the intComputers variable to hold the number of computers, you'll be using an integer to designate the number. Thus the prefix *int*. That's really all there is to it. It helps to keep you from thinking that maybe intComputers holds the names of the computers, which would be a string rather than an integer.

Why Hungarian? Because the first person to propose this type of variable naming was former Microsoft Distinguished Engineer Charles Simonyi, who happened to be Hungarian.

More Information

Using Hungarian Notation - Windows 2000 Scripting Guide

Top of page

A Constant Reminder

Another common saying (at least where we're from) is "if it looks like a duck and quacks like a duck, it's probably a duck." There is another type of element you'll see in scripts that looks like a variable and acts a little like a variable, but it's definitely *not* a variable. These are called constants.

There are a few differences between a constant and a variable. The first is that you have to actually say it's a constant. You do that by prefacing the constant with the word **Const**, like this:

```
Const x = "I'm a constant"
```

Another difference is that you can only put a literal value into a constant. That means you can't do something like this:

```
Const x = 1 + 1
```

You actually have to give it the value you want it to have:

```
Const x = 2
```

The biggest difference between a constant and a variable—now pay attention, this one messes people up all the time—is that you can't change the constant later in the script. Remember how we said the value in a variable can vary (making "variable" a good name for it)? Well, the value in a constant can't vary—it's constant (isn't that convenient?). Earlier we showed you a script where we set a variable and changed it later in the same script:

```
x = 1 + 1
Wscript.Echo x
x = 2 + 2
Wscript.Echo x
```

Now try it like this:

```
Const x = 2
Wscript.Echo x
x = 4
Wscript.Echo x
```

You'll get an error message on the third line (x = 4) because we said x was a constant and gave it a value (2), so we can't give it a different value later. You would typically use constants for values that you would never want to change.

Top of page

Advanced Constants

Here's an example that has some practical use. (Check this Hey, Scripting Guy article for a full explanation of how it works.) In this example, the script reads each line from a text file and displays the line. (Note that in order for this script to run, you must have a text file named Test.txt in your C:\Scripts folder.)

```
Const ForReading = 1
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objTextFile = objFSO.OpenTextFile _
    ("C:\Scripts\Test.txt", ForReading)
Do Until objTextFile.AtEndOfStream
    strTextLine = objTextFile.Readline
    Wscript.Echo strTextLine
Loop
```

Notice the very first line of the script. We've given the constant ForReading a value of 1. We won't get into methods and parameters and those types of things right now, we'll just say that in order to open a text file and read from it, you need to give the OpenTextFile method a value of 1. We put the 1 in the ForReading constant for two reason: so we know when we look at this script what the 1 is doing, and because we don't want this value to change. If we used a regular variable and changed this value to 2, suddenly our ForReading variable would open the text file for writing, which would cause us some headaches.

More Information

Constants – Windows 2000 Scripting Guide

Creating Constants – VBScript User's Guide

Top of page

VBScript Constants

Now, before the email starts pouring in, we want to explain that there are other types of constants. For example, you might see a script that uses something that looks like a variable because it was never put in a Const statement, but you still can't change the value. Try this script:

```
MsgBox "This is a message box", vbOKCancel
```

This will display a message box with OK and Cancel buttons. Try changing the value of vbOKCancel:

```
vbOKCancel = 5
MsgBox "This is a message box", vbOKCancel
```

You'll get an error message. Why? Because vbOKCancel is a constant, it just happens to be a constant we didn't have to define ourselves—VBScript took care of it for us. There is a standard set of constants that's been made available to us just because we're using VBScript. (Other scripting languages also have these types of constants.) These constants define things such as colors, days of the week, and (obviously) message box buttons. Because these are frequently-used constants, VBScript went ahead and defined them for us so we don't have to worry about it. (These constants are easy to recognize because they all begin with "vb", so it's best not to name any of your own variables or constants to start that way.)

Top of page

More Advanced Constants

Let's take a look at a script from a recent Office Space article:

```
Const wdLineStyleSingle = 1
Const wdLineStyleNone = 0

Set objWord = CreateObject("Word.Application")
objWord.Visible = True
Set objDoc = objWord.Documents.Add()
Set objSelection = objWord.Selection

i = 1

objDoc.Paragraphs(i).Range.Select
objSelection.ParagraphFormat.Borders.OutsideLineStyle = wdLineStyleSingle
objSelection.ParagraphFormat.Borders.Shadow = TRUE
objSelection.TypeText "This is the first paragraph."
objSelection.TypeParagraph()
i = i + 1

objDoc.Paragraphs(i).Range.Select
objSelection.ParagraphFormat.Borders.OutsideLineStyle = wdLineStyleNone
objSelection.TypeText "This is the second paragraph."
objSelection.TypeParagraph()
i = i + 1

objSelection.TypeText "This is the third paragraph."
objSelection.TypeParagraph()
i = i + 1
```

This script creates a Microsoft Word 2003 document and puts a border around the first paragraph. The part we're looking at today is the first two lines:

```
Const wdLineStyleSingle = 1
Const wdLineStyleNone = 0
```

As you can see from the script, each of these constants is assigned a value then used once in the script. A

question that people ask frequently is "Why not just use a variable?" There are a couple of reason for that.

The first, which we've already mentioned, is that we don't want these values to accidentally get changed anywhere in the script. wdLineStyleSingle must have a value of 1 and wdLineStyleNone must have a value of 0 in order for the script to work correctly.

The second reason, believe it or not, has to do with non-scripting languages. One of the things you learned in our first article was that scripting isn't as powerful as using a full-blown programming language such as Visual Basic .NET. One difference (of many) is that Visual Basic .NET can pull in predefined constants, whereas VBScript can't pull in much of anything. This has to do with linking to libraries and compiling and things like that, all of which you don't have to worry about because scripts aren't able to do these things.

So if you look at this section of the MSDN documentation and click on WdLineStyle at the top, you'll see a list of constants that can be used to define line styles in Word. You'll also notice that there are no values shown. If we were programming in Visual Basic .NET or even Visual Basic for Applications (VBA) we wouldn't care what the value is. It's predefined for us, we can use the constant wherever we need it. In VBScript, we need the actual values that these constants represent. You can find the list of values in the same Office Space article we got the script from. (Where that list came from is a topic for another day. For now just trust that it's accurate. That article did, after all, come from the Scripting Guys, so how could it be wrong?)

So, we used the constants in that table to define our line styles. Could we have just used the values and not defined constants at all? Yes. Could we have used variables rather than constants? Yes. Could we have named the variable or constant something different, such as LSSingle rather than wdLineStyleSingle? Yes. Why did we decide to create a constant, name it just like the documentation, and assign it a value? Well, for one thing your constants will be consistent across scripts. For another, once you've defined these constants in a script, you can search for them later on MSDN and find out what other constants are available, or you can find out what these are doing. So you don't have to follow this standard, but it really can make your life easier. We do it on the Script Center to make your lives easier. After all that's why we're here.

More Information

VBScript Constants – VBScript Language Reference

Microsoft Word VBA Language Reference

Top of page

Scripting Puzzles

That's enough for now. But to celebrate the start of our new beginner's series, we're starting up a new feature here on the Script Center. We're going to present you with a script that doesn't work, and challenge you to figure out what the problem is. The first puzzle relates to this article, you should be able to use what you've learned here to solve it (although it's a little tricky, you'll have to make sure you read carefully to figure it out). Learn more about the puzzles and try your hand at the first one. Good luck!

Top of page