

CLng

Converts a given expression into a Long type.

Syntax

```
Return = CLng( expression )
```

- Return: *Long*.
- Expression: *Numeric*.

Description

If you have a numeric expression that is too large to convert to an Integer, then you need to convert it to a variant subtype of Long with CLng. This function will round any fractional parts to the next whole number. Other integer related functions like Fix and Int only truncate anything to the right of a decimal point.

Practical Example

```
i=1073741824.678  
WScript.Echo "i=" & i  
WScript.Echo "CLng(i)=" & CLng(i) 'returns 1073741825
```

This short snippet illustrates what you can expect from CInt. Given a value of 1073741824.678, the function returns a value of 1073741825.

See Also

Fix
Int
CInt
CDBl
CSng.

ConnectObject

Connect to an object and access its events.

Syntax

```
object.ConnectObject( strObject,Prefix )
```

- **Object:** *A Wscript object.*
- **strObject:** *Object.* A COM or automation object that you want to connect to. This is required.
- **Prefix:** *String.* A string that will be used as a prefix for an event-related script function. This is required.

Description

When you need to work with an object's events, you need to use `ConnectObject` to establish a connection. An event in this context is something that *happens* as defined in the object such as *Start*, *End* or *Error*. When these events happen, they are said to have *fired* in COM-speak. Not every automation object supports or requires events. But if they do and you want to access them, you need to use `ConnectObject`.

Practical Example

```
Dim objController
Dim objRemote
strSrv="\\GODOT"
Const FINISHED = 2

Set objController=CreateObject("WSHController")
Set objRemote=objController.CreateScript("c:\adminage.vbs", strSrv)

WScript.ConnectObject objRemote, "Remote_"
objRemote.Execute

Do While objRemote.Status<>FINISHED
    WScript.Sleep 200
Loop

objRemote.Terminate
WScript.DisconnectObject objRemote

Function Remote_End()
    WScript.Echo "objRemote End Event fired"
End Function

Function Remote_Error()
    WScript.Echo "REMOTE ERROR!"
    WScript.Echo "Error Number: 0x" _
        & CStr(Hex(objRemote.Error.Number))
    WScript.Echo "Description: " & objRemote.Error.Description
    WScript.Echo "Line Number: " & objRemote.Error.Line
    objRemote.Terminate()
End Function

Function Remote_Start()
    WScript.echo "objRemote Start Event fired"
End Function
```

I won't go into great detail about this script. I'll cover the WshRemote object in more detail in Chapter 14. What I want to point out is that I've created an object called objRemote:

```
Set objRemote=objController.CreateScript("c:\adminage.vbs", strSrv)
```

I then establish a connection to the object with ConnectObject.

```
WScript.ConnectObject objRemote, "Remote_"
```

The first parameter is the name of the object. The second parameter is a prefix that is attached to the name of the object's events. For example, the WshRemote object has an event called Start. When that event fires, the script will execute any code associated with that event. I accomplish this by creating a function with the prefix and the event name.

```
Function Remote_Start()  
    WScript.echo "objRemote Start Event fired"  
End Function
```

The sample script provides functions for the three events that this object can fire.

See Also

GetObject

CreateObject

DisconnectObject.

CreateObject

Create an automation object.

Syntax

```
Return = CreateObject(servername.typename [,location] )
```

- Return: *Automation Object*.
- servername: *String*. The name of the COM server providing the object. This is required.
- Typename: *String* The type or class of the object to create. This is required.
- Location: *String* The name of the server or computer where this object will be created.

Description

This function creates a reference to an automation object, also sometimes referred to as a COM (Component Object Model) object. Once you have a reference to an object, you can access its properties and methods directly using a dotted format.

In order to create an object, the underlying automation files, typically a dll, must be installed and registered on your computer. Once registered, you can find these objects in the registry under HKEY_CLASSES_ROOT. The combination of *servername* and *typename* is typically referred to as the object's *ProgID*. You must specify this name when creating an object. Even though there is an optional parameter for a remote server name, not all COM objects support this parameter. In most administrative scripting you will not use this parameter.

Practical Example

```
Dim objNetwork
Set objNetwork=CreateObject("WScript.Network")
WScript.Echo "Your computer name is " & objNetwork.ComputerName
```

This is a pretty typical use of `CreateObject`. The ProgID of the COM object is `WScript.Network`. The script creates a reference object called `objNetwork`. Notice I am using *Set* because `objNetwork` will be an object and not just a variable.

Once this object reference is created, I can access the *ComputerName* property of the `WScript.Network` object with dotted notation.

```
WScript.Echo "Your computer name is " & objNetwork.ComputerName
```

See Also

[GetObject](#)

[ConnectObject](#)

[DisconnectObject](#)

CSng

Converts a numeric expression to a `Single` type.

Syntax

```
Return = CSng( expression )
```

- Return: *Single*.
- Expression: *Numeric*. Any valid numeric expression.

Description

Use `CSng` when your script for calls precise calculation. This function will convert any given numeric expression into a `Single` subtype. This subtype is essentially a number of 8 characters, including a decimal point.

Practical Example

```
i=1234.567890123
WScript.Echo "      i=" & i
WScript.Echo "CSng(i)=" & CSng(i) 'Returns 1234.568
```

You'll rarely need this function in administrative scripts. This short demo illustrates what type of result you can expect using this function.

See Also

Fix

Int

CInt

CDBl

CLng.

CStr

Converts a given expression into a string.

Syntax

```
Return = CStr( expression )
```

- Return: *String*.
- Expression: *Any*. You can enter any non-string expression.

Description

For the most part VBScript treats everything as a variant type. When you need to force an expression to be treated as a string, you would use CStr. The return value of this function depends on what type of expression it is evaluating.

- Numeric: CStr will return a string containing the number. This is an important distinction as illustrated in the example below.
- Boolean: If the expression is True or False, CStr will return the string of either True or False.
- Date: CStr will return a string version of the date using the short-date format as determined by your regional settings.

- Empty: If CStr tries to evaluate an expression like "", nothing will be returned no error will be generated.
- Null: If CStr tries to evaluate a Null expression, a run time error will be generated.
- Error: If CStr tries to evaluate an Error object, it will return the error number as a string. From a practical perspective this is no different than using Err.Number.

Practical Example

```
a=5
b=7
WScript.Echo a+b 'returns 12
WScript.Echo CStr(a)+CStr(b) 'returns 57
```

In this example I'm demonstrating what happens when you convert a number to a string. When I use the + sign to add variables a and b, I get the expected result of 12. However, when variables a and b are converted to strings, they lose their mathematical functionality. Thus when I "add" them in the last line they are concatenated and the result is 57.

Let's look at another example.

```
d=CDate("Feb 22,1961 19:53:45")
WScript.Echo CStr(d) 'returns 2/22/1961 7:53:45 PM
```

Here I'm creating a variable, d, that is of type Date. When the variable is converted to a string it is returned as the short-date equivalent.

See Also

CBool
CByte
CCur
CDate
CDBl
CInt
CLng
CSng

Date

Returns the current system date.

Syntax

Return = Date

- Return: *Date*.

Description

This simple function returns the current system date in short-date format based on your regional settings.

Practical Example

```
today=Date
strFile=Replace(today,"/","") & ".txt"

Set objFSO=CreateObject("Scripting.FileSystemObject")
Set objFile=objFSO.CreateTextFile(strFile,True)

Set objWMI=GetObject("WinMgmts:{(Security)}//")
Set objRef=objWMI.ExecQuery("Select * from win32_NTEventLogFile")

For Each file In objRef
    objFile.WriteLine file.LogFileName
    objFile.WriteLine " " & file.name
    objFile.WriteLine " Size:" & file.FileSize & " bytes"
    objFile.WriteLine " Records:" & file.NumberOfRecords
Next

objFile.Close
WScript.Echo "Event log status logged to " & strFile
```

This script could be used to capture event log information for the local computer such as the file name, size and number of entries. The information is written to a log file. The log file name is based on the current date. At the beginning of the script I set the variable *today* to return value of *Date*.

today=Date

This will return a value like 12/20/2006. But I can use the *Replace* function to strip out the / characters and create a filename like 12202006.txt.

```
strFile=Replace(today,"/","") & ".txt"
```

This filename variable can be used by the FileSystemObject to create a new text file. The rest of the script pulls information about NT Eventlogs from WMI (Windows Management Instrumentation).

See Also

Other date and time related functions are Time, CDate and Now.