Microsoft | TechNet

New-Object

Updated: April 21, 2010

Applies To: Windows PowerShell 2.0

Creates an instance of a Microsoft .NET Framework or COM object.

Syntax

```
New-Object -ComObject <string> [-Strict] [-Property <hashtable>] [<CommonParameters>]

New-Object [-TypeName] <string> [[-ArgumentList] <Object[]>] [-Property <hashtable>] [<CommonP
arameters>]
```

Description

The New-Object cmdlet creates an instance of a .NET Framework or COM object.

You can specify either the type of a .NET Framework class or a ProgID of a COM object. By default, you type the fully qualified name of a .NET Framework class and the cmdlet returns a reference to an instance of that class. To create an instance of a COM object, use the ComObject parameter and specify the ProgID of the object as its value.

Parameters

-ArgumentList <Object[]>

Specifies a list of arguments to pass to the constructor of the .NET Framework class. Separate elements in the list by using commas (,). The alias for ArgumentList is Args.

Required?	false
Position?	2
Default Value	none
Accept Pipeline Input?	false
Accept Wildcard Characters?	false

-ComObject <string>

Specifies the programmatic identifier (ProgID) of the COM object.

Required?	true
Position?	named
Default Value	None
Accept Pipeline Input?	false
Accept Wildcard Characters?	false

-Property <hashtable>

Sets property values and invokes methods of the new object.

Enter a hash table in which the keys are the names of properties or methods and the values are property values or method arguments. New-Object creates the object and sets each property value and invokes each method in the order that they appear in the hash table.

If the new object is derived from the PSObject class, and you specify a property that does not exist on the object, New-Object adds the specified property to the object as a NoteProperty. If the object is not a PSObject, the command generates a non-terminating error.

Required?	false
Position?	named
Default Value	none
Accept Pipeline Input?	false
Accept Wildcard Characters?	false

-Strict

Specifies that an error should be raised if the COM object that you attempt to create uses an interop assembly. This enables you to distinguish actual COM objects from .NET Framework objects with COM-callable wrappers.

Required?	false
Position?	named
Default Value	none

Accept Pipeline Input?	false
Accept Wildcard Characters?	false

-TypeName <string>

Specifies the fully qualified name of the .NET Framework class. You cannot specify both the TypeName parameter and the ComObject parameter.

Required?	true
Position?	1
Default Value	none
Accept Pipeline Input?	false
Accept Wildcard Characters?	false

<CommonParameters>

This command supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, OutBuffer, OutVariable, WarningAction, and WarningVariable. For more information, see about_CommonParameters[1].

Inputs and Outputs

The input type is the type of the objects that you can pipe to the cmdlet. The return type is the type of the objects that the cmdlet returns.

Inputs	None You cannot pipe input to this cmdlet.
Outputs	Object New-Object returns the object that is created.

Notes

New-Object provides the most commonly-used functionality of the VBScript CreateObject function. A statement like Set objShell = CreateObject("Shell.Application") in VBScript can be translated to $objShell = new-object -comobject "Shell.Application" in Windows PowerShell.

New-Object expands upon the functionality available in the Windows Script Host environment by making it easy to work with .NET Framework objects from the command line and within scripts.

Example 1

```
C:\PS>new-object -typename System.Version -argumentlist "1.2.3.4"

Major  Minor  Build  Revision
-----  -----  -----  --------
1      2      3      4
```

Description

This command creates a System.Version object using the string "1.2.3.4" as the constructor.

Example 2

```
C:\PS>$ie = new-object -comobject InternetExplorer.Application -property @{navigate2="www.micr
osoft.com"; visible = $true}
```

Description

This command creates an instance of the COM object that represents the Internet Explorer application. It uses the Property parameter to call the Navigate2 method and to set the Visible property of the object to $true to make the application visible.

This command is the equivalent of the following:

$ie = new-object -comobject InternetExplorer.Application

$ie.navigate2("www.microsoft.com")

$ie.visible = $true

Example 3

```
C:\PS>$a=new-object -comobject Word.Application -strict -property @{visible=$true}

New-Object : The object written to the pipeline is an instance of the type
"Microsoft.Office.Interop.Word.ApplicationClass" from the component's prima
ry interop assembly. If this type exposes different members than the IDispa
tch members, scripts written to work with this object might not work if the
 primary interop assembly is not installed.
At line:1 char:14
+ $a=New-Object  <<<< -COM Word.Application -Strict; $a.visible=$true
```

Description

This command demonstrates that specifying the Strict parameter causes the New-Object cmdlet to generate a non-terminating error when the COM object that is created uses an interop assembly.

Example 4

```
C:\PS>$objshell = new-object -comobject "Shell.Application"

C:\PS> $objshell | get-member

C:\PS> $objshell.ToggleDesktop()
```

Description

The command uses the ComObject parameter to create a COM object with the "Shell.Application" ProgID. It stores the resulting object in the $objShell variable.

The second command pipes the $objShell variable to the Get-Member cmdlet, which displays the properties and methods of the COM object.

The third command calls the ToggleDesktop method of the object to minimize the open windows on your desktop.

See Also

Concepts

Compare-Object[2]
Select-Object[3]
Sort-Object[4]
ForEach-Object[5]
Group-Object[6]
Measure-Object[7]
Tee-Object[8]
Where-Object[9]

Links Table

[1]*http://technet.microsoft.com/en-US/library/dd315352.aspx*

[2]*http://technet.microsoft.com/en-US/library/dd347568.aspx*

[3]*http://technet.microsoft.com/en-US/library/dd315291.aspx*

[4]*http://technet.microsoft.com/en-US/library/dd347688.aspx*

[5]*http://technet.microsoft.com/en-US/library/dd347608.aspx*

[6]*http://technet.microsoft.com/en-US/library/dd347561.aspx*

[7]*http://technet.microsoft.com/en-US/library/dd315251.aspx*

[8]*http://technet.microsoft.com/en-US/library/dd347705.aspx*

[9]*http://technet.microsoft.com/en-US/library/dd315399.aspx*

Community Content

Response: Creating Generic Types

This is great, Keith. Can you please explain to readers when\why they would use a generic type? It would be great to hear about the types of problems this solves.

7/29/2010
JuneB-MSFT

Creating Generic Types

```
PS> $list = New-Object 'System.Collections.Generic.List[string]'
PS> $list.Add('foo')
PS> $list

foo

PS> $d = New-Object 'System.Collections.Generic.Dictionary[string,datetime]'
PS> $d.Add('moonshot', [datetime]'7/20/1969')
PS> $d['moonshot']

Sunday, July 20, 1969 12:00:00 AM
```

1/21/2010
Keith Hill MVP