



Full credit is given to all the above companies including the Operating System that this PDF file was generated!

Windows PowerShell Get-Help on Cmdlet 'New-Object'

PS:\>Get-HELP New-Object -Full

NAME

New-Object

SYNOPSIS

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

SYNTAX

```
New-Object [-TypeName] <System.String> [[-ArgumentList] <System.Object[]>] [-Property <System.Collections.IDictionary>] [<CommonParameters>]
```

```
New-Object [-ComObject] <System.String> [-Property <System.Collections.IDictionary>] [-Strict] [<CommonParameters>]
```

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 <System.Object[]>

Specifies an array of arguments to pass to the constructor of the .NET Framework class. If the constructor takes a single parameter that is an array, you must

wrap that parameter inside another array. For example:

```
`$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate -ArgumentList ($bytes)`
```

For more information about the behavior of ArgumentList , see about_Splatting
(../Microsoft.PowerShell.Core/About/about_Splatting.md#splatting-with-arrays).

The alias for ArgumentList is Args .

Required? false

Position? 1

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-ComObject <System.String>

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

Required? true

Position? 0

Default value None

Accept pipeline input? False

Accept wildcard characters? false

-Property <System.Collections.IDictionary>

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 <System.Management.Automation.SwitchParameter>

Indicates that the cmdlet generates a non-terminating error when a COM object that you attempt to create uses an interop assembly. This feature distinguishes

actual COM objects from .NET Framework objects with COM-callable wrappers.

Required? false

Position? named

Default value False

Accept pipeline input? False

Accept wildcard characters? false

-TypeName <System.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? 0

Default value	None
Accept pipeline input?	False
Accept wildcard characters?	false

<CommonParameters>

This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (<https://go.microsoft.com/fwlink/?LinkId=113216>).

INPUTS

None

You can't pipe objects to this cmdlet.

OUTPUTS

System.Object

This cmdlet returns the object that it creates.

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 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: Create a System.Version object -----

```
New-Object -TypeName System.Version -ArgumentList "1.2.3.4"
```

Major Minor Build Revision

1 2 3 4

----- Example 2: Create an Internet Explorer COM object -----

```
$IE1 = New-Object -COMObject InternetExplorer.Application -Property @{Navigate2="www.microsoft.com"; Visible = $True}
```

The following command gets the same results as the example above.

```
$IE2 = New-Object -COMObject InternetExplorer.Application`  
$IE2.Navigate2("www.microsoft.com")`  
$IE2.Visible = $True`
```

Example 3: Use the Strict parameter to generate a non-terminating error

```
$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 primary interop assembly. If this type exposes different members than the IDispatch 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
```

--- Example 4: Create a COM object to manage Windows desktop ---

```

$Objshell = New-Object -COMObject "Shell.Application"
$Objshell | Get-Member
$Objshell.ToggleDesktop()

TypeName: System.__ComObject#{866738b9-6cf2-4de8-8767-f794ebe74f4e}

Name          MemberType Definition
----          -----
AddToRecent   Method    void AddToRecent (Variant, string)
BrowseForFolder Method   Folder BrowseForFolder (int, string, int, Variant)
CanStartStopService Method  Variant CanStartStopService (string)
CascadeWindows  Method   void CascadeWindows ()
ControlPanellItem Method  void ControlPanellItem (string)
EjectPC        Method   void EjectPC ()
Explore         Method   void Explore (Variant)
ExplorerPolicy  Method   Variant ExplorerPolicy (string)
FileRun         Method   void FileRun ()
FindComputer   Method   void FindComputer ()
FindFiles       Method   void FindFiles ()
FindPrinter    Method   void FindPrinter (string, string, string)
GetSetting      Method   bool GetSetting (int)
GetSystemInformation Method Variant GetSystemInformation (string)
Help            Method   void Help ()
IsRestricted    Method   int IsRestricted (string, string)
IsServiceRunning Method Variant IsServiceRunning (string)
MinimizeAll     Method   void MinimizeAll ()
NameSpace        Method   Folder NameSpace (Variant)
Open            Method   void Open (Variant)
RefreshMenu     Method   void RefreshMenu ()
ServiceStart    Method   Variant ServiceStart (string, Variant)
ServiceStop     Method   Variant ServiceStop (string, Variant)
SetTime          Method   void SetTime ()
ShellExecute    Method   void ShellExecute (string, Variant, Variant, Variant, Variant)

```

```

ShowBrowserBar      Method   Variant ShowBrowserBar (string, Variant)
ShutdownWindows    Method   void ShutdownWindows ()
Suspend            Method   void Suspend ()
TileHorizontally   Method   void TileHorizontally ()
TileVertically     Method   void TileVertically ()
ToggleDesktop     Method   void ToggleDesktop ()
TrayProperties     Method   void TrayProperties ()
UndoMinimizeALL   Method   void UndoMinimizeALL ()
Windows            Method   IDispatch Windows ()
WindowsSecurity    Method   void WindowsSecurity ()
WindowSwitcher    Method   void WindowSwitcher ()
Application        Property  IDispatch Application () {get}
Parent             Property  IDispatch Parent () {get}

```

----- Example 5: Pass multiple arguments to a constructor -----

```

$array = @('One', 'Two', 'Three')
$parameters = @{
    TypeName = 'System.Collections.Generic.HashSet[string]'
    ArgumentList = ([string[]]$array, [System.StringComparer]::OrdinalIgnoreCase)
}
$set = New-Object @parameters

```

PowerShell binds each member of the array to a parameter of the constructor.

> [!NOTE] > This example uses parameter splatting for readability. For more information, see > about_Splatting
 (../Microsoft.PowerShell.Core/About/about_Splatting.md).

Example 6: Calling a constructor that takes an array as a single parameter

```

$array = @('One', 'Two', 'Three')
# This command throws an exception.
$set = New-Object -TypeName 'System.Collections.Generic.HashSet[string]' -ArgumentList $array

```

```
# This command succeeds.  
  
$set = New-Object -TypeName 'System.Collections.Generic.HashSet[string]' -ArgumentList ([string[]]$array)  
  
$set
```

New-Object : Cannot find an overload for "HashSet`1" and the argument count: "3".

At line:1 char:8

```
+ $set = New-Object -TypeName 'System.Collections.Generic.HashSet[stra ...  
+ ~~~~~  
+ CategoryInfo          : InvalidOperationException: () [New-Object], MethodException  
+ FullyQualifiedErrorId : ConstructorInvokedThrowException,Microsoft.PowerShell.Commands.NewObjectCommand
```

One

Two

Three

The first attempt to create the object in this example fails. PowerShell attempted to bind the three members of `'\$array` to parameters of the constructor but the

constructor does not take three parameter. Wrapping `'\$array` in another array prevents PowerShell from attempting to bind the three members of `'\$array` to parameters of the constructor.

RELATED LINKS

Online

Version:

https://learn.microsoft.com/powershell/module/microsoft.powershell.utility/new-object?view=powershell-5.1&WT.mc_id=ps-gethelp

about_Object_Creation

Compare-Object

Group-Object

Measure-Object

Select-Object

Sort-Object

Tee-Object

