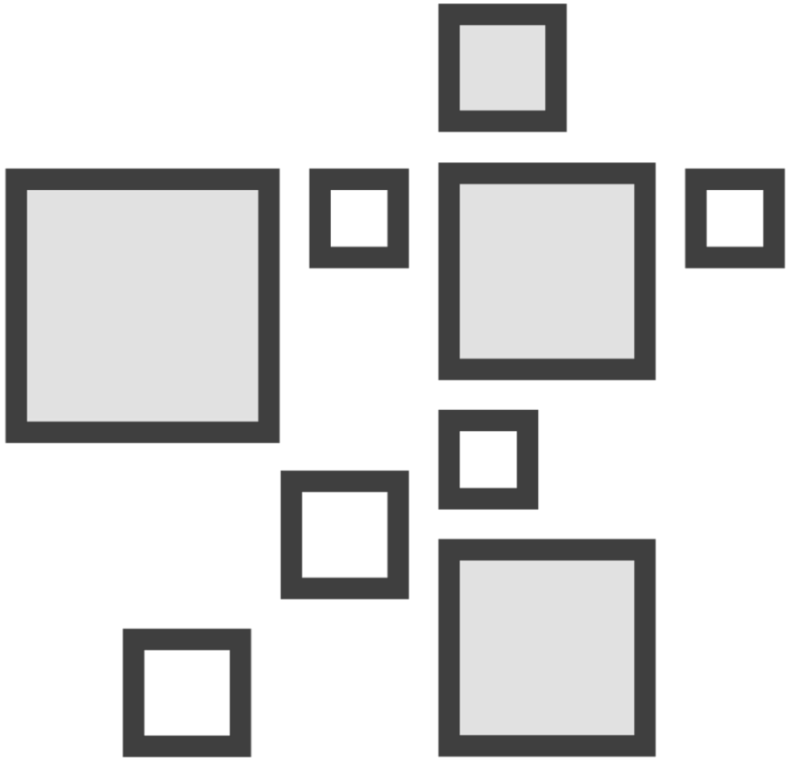
Working with Objects in Windows PowerShell

Jeff Hicks
Author | Teacher

https://jdhitsolutions.github.io

Object

A software representation of some “thing” you want to manage or work with.



An object is “black box”

We don’t care how it is constructed

We want to know how to use it

Everything in PowerShell is some type of object

Object Members

Property

Method

Event

Common Object Types

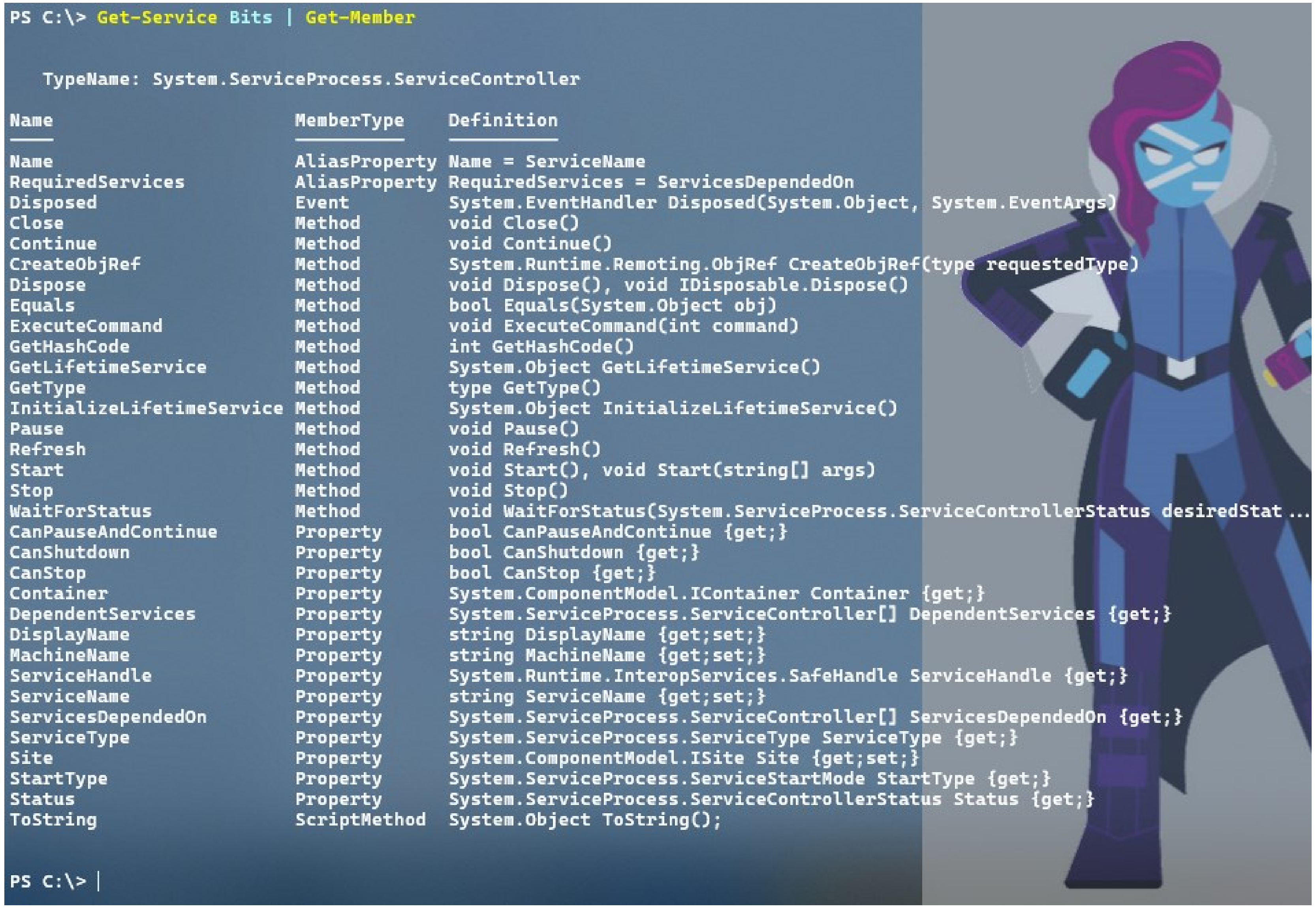
String

Int32

DateTime

Boolean

Get-Member



```
PS C:\> Get-Service Bits | Get-Member


   TypeName: System.ServiceProcess.ServiceController

Name                      MemberType    Definition
----                      ----------    ----------
Name                      AliasProperty Name = ServiceName
RequiredServices          AliasProperty RequiredServices = ServicesDependedOn
Disposed                  Event         System.EventHandler Disposed(System.Object, System.EventArgs)
Close                     Method        void Close()
Continue                  Method        void Continue()
CreateObjRef              Method        System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose                   Method        void Dispose(), void IDisposable.Dispose()
Equals                    Method        bool Equals(System.Object obj)
ExecuteCommand            Method        void ExecuteCommand(int command)
GetHashCode               Method        int GetHashCode()
GetLifetimeService        Method        System.Object GetLifetimeService()
GetType                   Method        type GetType()
InitializeLifetimeService Method        System.Object InitializeLifetimeService()
Pause                     Method        void Pause()
Refresh                   Method        void Refresh()
Start                     Method        void Start(), void Start(string[] args)
Stop                      Method        void Stop()
WaitForStatus             Method        void WaitForStatus(System.ServiceProcess.ServiceControllerStatus desiredStat ...
CanPauseAndContinue       Property      bool CanPauseAndContinue {get;}
CanShutdown               Property      bool CanShutdown {get;}
CanStop                   Property      bool CanStop {get;}
Container                 Property      System.ComponentModel.IContainer Container {get;}
DependentServices         Property      System.ServiceProcess.ServiceController[] DependentServices {get;}
DisplayName               Property      string DisplayName {get;set;}
MachineName               Property      string MachineName {get;set;}
ServiceHandle             Property      System.Runtime.InteropServices.SafeHandle ServiceHandle {get;}
ServiceName               Property      string ServiceName {get;set;}
ServicesDependedOn        Property      System.ServiceProcess.ServiceController[] ServicesDependedOn {get;}
ServiceType               Property      System.ServiceProcess.ServiceType ServiceType {get;}
Site                      Property      System.ComponentModel.ISite Site {get;set;}
StartType                 Property      System.ServiceProcess.ServiceStartMode StartType {get;}
Status                    Property      System.ServiceProcess.ServiceControllerStatus Status {get;}
ToString                  ScriptMethod  System.Object ToString();


PS C:\> |
```

Type Operators

-Is

-As

```
PS C:\> $s = "foo"
```

Object Notation

Reference an object via a variable

```
PS C:\> $s = "foo"
PS C:\> $s.length
```

Object Notation

Reference an object via a variable
Separate member from the object with a period

```
PS C:\> $s = "foo"
PS C:\> $s.length
3
```

Object Notation

Reference an object via a variable
Separate member from the object with a period

```
PS C:\> $s = "foo"
PS C:\> $s.length
3
PS C:\> $s | Get-Member
```

Object Notation

Reference an object via a variable
Separate member from the object with a period
Use Get-Member to discover an object’s properties and methods

```
PS C:\> $s.ToUpper()
```

Object Methods

Methods always use parentheses, even if no parameters

```
PS C:\> $s.ToUpper()
FOO
```

Object Methods

Methods always use parentheses, even if no parameters
Doesn’t change the variable

```
PS C:\> $s.padleft

OverloadDefinitions
-------------------
string PadLeft(int totalWidth)
string PadLeft(int totalWidth, char paddingChar)
```

Object Methods

Discover method parameters

```
PS C:\> $s.PadLeft(10,"-")
-------foo
```

Object Methods

Pass method parameters

```
PS C:\>  $s.toUpper().PadLeft(10,"-")
-------FOO
```

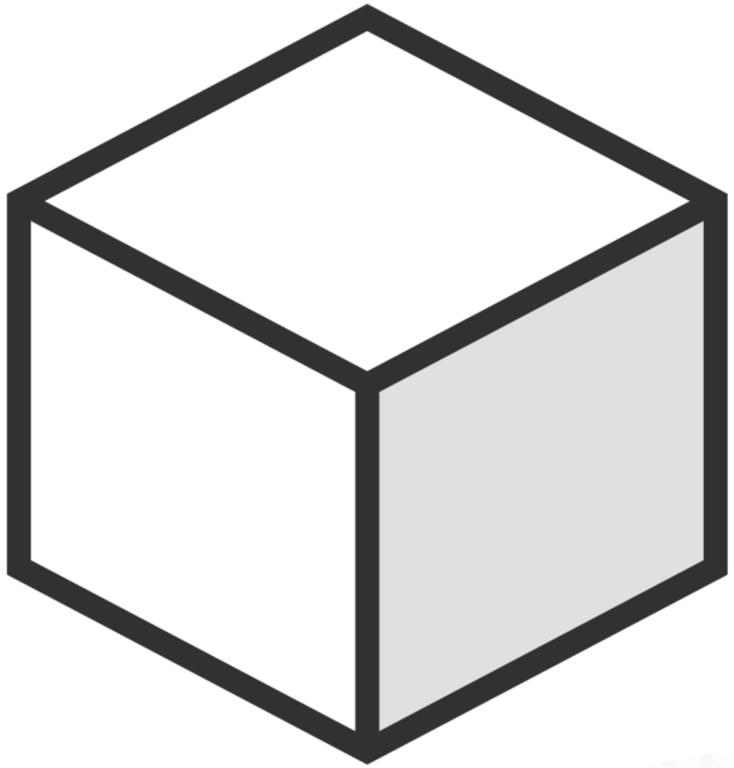
Object Methods

You can get creative

Don’t focus on learning or using an objects methods. Look for cmdlets that implement the method.

Working with Objects

Custom Properties and Objects



Always think “objects”, not “text”

Create custom properties you need

Create custom objects you need

Creating Custom Properties

```
Get-ChildItem -path C:\data -file -recurse |
Select-Object -property Fullname,Name,LastWriteTime,
@{Name = "Size"; Expression = {$_.Length}},
@{Name = "ComputerName"; Expression = {$env:COMPUTERNAME}},
@{Name = "Audit"; Expression = {(Get-Date -format g)}} |
Export-CSV c:\work\data.csv
```

This also creates a custom object

Creating Custom Objects

```
$ps = Get-Process
$os = Get-CimInstance win32_OperatingSystem
$svc = Get-Service | Where-Object {$_.status -eq 'running'}

$h = @{
 ComputerName = $env:computername
 Version      = $PSVersionTable.PSVersion
 ProcessCount = $ps.count
 ServiceCount = $svc.count
 Uptime       = New-TimeSpan -start $os.LastBootUpTime -end (Get-Date)
}

New-Object -TypeName PSObject -Property $h
```

Use the [PSCustomObject] shortcut

Demo

Custom Properties and Objects

Key Take-Aways

Always be thinking about objects

Look for ways to leverage dotted notation

When you start scripting, think “Rich objects in the pipeline.”

There are advanced scripting techniques

- Add-Member
- Update-TypeData
- Update-FormatData

Practice and play