# The DSC Book



PowerShell.org

www.dbooks.org

# The DSC Book

PowerShell.org

This project can be followed at:

https://www.penflip.com/powershellorg/the-dsc-book

©2018 PowerShell.org

# Contents



www.dbooks.org

# 1 The DSC Book

by Don Jones
and Steve Murawski
with contributions by Stephen Owen
cover design by Nathan Vonnahme

Visit Penflip.com/powershellorg to check for newer editions of this ebook.

This guide is released under the Creative Commons Attribution-NoDerivs 3.0 Unported License. The authors encourage you to redistribute this file as widely as possible, but ask that you do not modify the document.

---

**This ebook is no longer being maintained**. It was written for PowerShell v4.0, and is still partially applicable to the release of DSC included in PowerShell v5.0. However, this particular book will not be updated for v5. Please visit http://powershell.org/wp/ebooks for other free PowerShell.org ebooks.

---

## About the Authors

Principal writing in this book was by Don Jones, President and CEO of PowerShell.org, Inc. and a multi-year recipient of Microsoft's MVP Award.

Additional writing, background information, and tech-checking were by Steve Murawski, possibly one of the earliest production adopters of DSC through his job at StackExchange, and one of Don's fellow PowerShell MVPs.

Several examples have been adapted, with permission, from http://foxdeploy.azurewebsites.net/2014/03/10/desired-state-configuration-what-it-is-and-why-you-should-care/ and other excellent posts by Stephen Owen.

It's important for you to know that this guide is *very much a work in progress*. We appreciate feedback (use the PowerShell Q&A forum at PowerShell.org), and welcome additional contributors. Because we're treating this book as an open-source project, you may be reading it before a complete tech-check has been completed - so we appreciate your patience!

# 2 The Argument for DSC

Desired State Configuration is Microsoft's technology, introduced in Windows Management Framework 4.0, for declarative configuration of systems. At the risk of oversimplifying a bit, DSC lets you create specially formatted text files that describe how a system should be configured. You then deploy those files to the actual systems, and they magically configure themselves as directed. At the top level, DSC isn't programming - it's just listing how you want a system to look.

Microsoft is actually a little late to the game on declarative configuration. The Linux and Unix world has had tools like Chef and Puppet, which at their very core perform a similar task, for a long time. What's important is that DSC isn't an add-on, it's a core part of the Windows OS, now. And while the toolset for DSC isn't yet as mature as tools like Chef and Puppet, DSC provides an inbuilt foundation that, in its v1 implementation, is incredibly robust and extensible.

DSC is the last major piece of functionality outlined in PowerShell inventor Jeffrey Snover's *Monad Manifesto*, a document in which he envisioned PowerShell itself, PowerShell Remoting, and a declarative configuration model. The implementation of that Manifesto has been happening since 2006, when PowerShell v1 was released to the public. For the most part (as you'll read in this guide), that implementation has been done in a way that's extremely cross-platform friendly. PowerShell Remoting, and even DSC, are based on open, third-party standards. With that standardized foundation, Microsoft can let other folks focus on tooling, knowing that those tools don't have to be Windows-dedicated since other operating systems can utilize these same technologies.

DSC is important - crucially important - to anyone who works with Windows servers. Even if that isn't the only OS in your environment, DSC is massively important, because it's going to gradually become *the only means by which you manage Windows-based servers*. That's a big statement, and it deserves some explanation.

## The Evolution of Administration

In its beginning, Windows was administered mostly by graphical tools, and a few command-line ones, communicating directly with services on local or remote computers. You clicked a button in a dialog box, and the tool would talk to the Security Accounts Manager (SAM) and create a user account.

As environments grew large and IT teams smaller, folks needed to automate many of those processes. Graphical tools are inherently difficult to automate; the first phase in Snover's *Monad Manifesto* was a command-line shell that provided a consistent command-line language for automating administration. Command-line tools are inherently easy to automate, because they can be sequenced in simple batch or script files. A *consistent* and *all-encompassing* command-line language was something Windows



www.dbooks.org

had never had; management pressures within Microsoft were brought to bear to make PowerShell as all-encompassing as possible (an investment still underway). Guidelines were put into place to make it as consistent as possible (with largely positive results).

It's important to note that commands continued to communicate directly with services. Run **New-ADUser** , for example, and the command communicates directly with a service running on a domain controller to create the new user.

Scale-out was the next phase of the *Manifesto*. A command-line automation language is fine, but if you have to run around typing commands on individual computers, you still have an upper limit on efficiency. The ability to push commands to remote computers, in parallel, over a standardized communication protocol, was needed. PowerShell v2 delivered that in its Remoting feature. Now, one computer could easily tell a thousand others what to do, more or less at once. Remoting also reduced some of the burden on command developers. They could now assume that their commands always ran locally, on whatever computer was running their technology, and let PowerShell handle network communications. Not having to deal with communications made it significantly easier for command developers to pump out more commands, increasing PowerShell's technological "coverage."

But to this point, systems administrators still ran commands, and commands talked to services. Everything was very *imperative*: computers did what admins told them to do, more or less right then. If the environment changed, admins had to construct new scripts to implement the new changes. Admins then had to go back and make sure the changes *had been implemented* correctly, and continually re-check to make sure things *stayed that way*. If something got mis-configured, the fix was often a fallback to manual reconfiguration. You couldn't just run the same script, since it was designed to take a system from Point A to Point B, not from Point-A-Except-For-This-One-Thing to Point B.

DSC represents a significant break in administration, because it asks administrators to *no longer communicate directly with services on computers*. That means no graphical tools that talk directly to a service. It also means no running commands that talk directly to services. In other words, DSC asks administrators to *not actually configure anything themselves*.

Instead, DSC asks administrators to describe, in fairly simple text files, how they would like a computer to be configured. The computer, in turn, reads that text file, and configures itself accordingly. The configuration process is granular, so that if one item gets mis-configured, the computer can still bring itself "into compliance" by fixing that one thing.

DSC couldn't have existed without PowerShell, because PowerShell is what gives DSC the ability to implement changes. PowerShell is the one common language that all Microsoft products speak (to some degree or another, with constant improvement). In other words, without PowerShell, Microsoft products wouldn't have consistent enough automation capabilities for DSC to exist. Now that DSC does exist, administrators can actually focus less on PowerShell, because they'll be separated from it by a couple of less-complex layers.

Imagine a graphical tool where you click through a wizard to add a new user to your environment. Only that tool doesn't actually talk to a domain controller to create the user account, nor does it talk to a file server to create their home directory. Instead, the tool merely modifies a couple of text files that contain the configuration for a cluster of file servers, and for your domain. The file servers and domain controllers periodically re-read those files, and implement whatever they say. So, you click a button, and a few minutes later the user, and their home directory, magically exists. It's

automated, but you didn't need to do any programming. The configuration files are where you, as an administrator, make your changes.

Growing the environment becomes simpler. "Hey, you are going to be a new domain controller. Go look at configuration file DCX34766578348 and make yourself look like what it says." Poof, done.

DSC represents a massive change in how Windows administrators think about their entire environment. Provided every configuration setting can be boiled down to a DSC setting - which will be true over time - then "administration" will essentially become "intelligent editing of text files." Pretty powerful stuff.

www.dbooks.org

# 3 DSC Overview and Requirements

Desired State Configuration (DSC) was first introduced as part of Windows Management Framework (WMF) 4.0, which is preinstalled in Windows 8.1 and Windows Server 2012 R2, and is available for Windows 7, Windows Server 2008 R2, and Windows Server 2012. Because Windows 8.1 is a free upgrade to Windows 8, WMF 4 is not available for Windows 8.

**You must have WMF 4.0 on a computer if you plan to author configurations there. You must also have WMF 4.0 on any computer you plan to manage via DSC. Every computer involved in the entire DSC conversation must have WMF 4.0 installed. Period.** Check $PSVersionTable in PowerShell if you're not sure what version is installed on a computer.

To figure out what DSC is and does, it's useful to compare it to Group Policy. There are significant differences between the two, but at a high level they both set out to accomplish something similar. With Group Policy, you create a declarative configuration file called a Group Policy object (GPO). That file lists a bunch of configuration items that you want to be in effect on one or more computers. You target the GPO by linking it to domain sites, organizational units, and so on. Targeted machines pull, or download, the entire GPO from domain controllers. The machines use client-side code to read the GPO and implement whatever it says to do. They periodically re-check for an updated GPO, too.

DSC is similar... but not exactly the same. For one, it has no dependencies whatsoever on Active Directory Domain Services (ADDS). It's also a lot more flexible and more easily extended. A comparison is perhaps a good way to get a feel for what DSC is all about:

| Feature | Group Policy |
|---|---|
| Configuration specification | GPO file |
| Targeting machines | Link GPO to sites, OUs, etc. |
| Configuration implemented by | Client-side OS components |
| Extend the things that can be configured | Client-side GP extensions - usually written in native code, somewhat diffi |
| Primary configuration target | Windows registry |
| Persistence | Settings "disappear" and re-applied each time for most GPO settings |
| Number of configurations | As many GPOs as you want |

With DSC, you start by writing a *configuration script* in Windows PowerShell. This script doesn't

*do* anything. That is, it doesn't install stuff, configure stuff, or anything else. It simply lists the things you want configured, and how you want them configured. The configuration also specifies the machines that it applies to. When you run the configuration, PowerShell produces a Managed Object Format (MOF) file for each targeted machine, or *node*.

That's an important thing to call out: You (step 1) write a configuration script in PowerShell. Then you (step 2) run that script, and the result is one or more MOF files. If your configuration is written to target multiple nodes, you'll get a MOF file for each one. MOF stands for *Managed Object Format*, and it's basically a specially formatted text file. Then, (step 3), the MOF files are somehow conveyed to the machines they're meant for, and (step 4) those machines start configuring themselves to match what the MOF says.

In terms of conveying the MOF files to their target machines, there are two ways to do so: *push* mode is a more-or-less manual file copy, performed over PowerShell's Windows Remote Management (WinRM) remoting protocol; the *pull* mode configures nodes to check in to a special web server and grab their MOF files. Pull mode is a lot like the way Group Policy works, except that it doesn't use a domain controller. Pull mode can also be configured to pull MOF files from a file server by using Server Message Blocks (SMB; Windows' native file-sharing protocol).

Once a node has its MOF file (and it's only allowed to have one; that's another difference from Group Policy, where you can target several GPOs to a single machine), it starts reading through the configuration. Each section in the configuration uses a *DSC resource* to actually implement the configuration. For example, if the configuration includes some kind of registry specification, then the registry DSC resource is called upon to actually check the registry and make the change if necessary.

You do have to deploy those DSC resources to your target nodes. In push mode, that's a manual task. In pull mode, nodes can realize that they're missing a resource needed by their configuration, and grab the necessary resource from the pull server (if you've put it there). For that reason, pull mode is the most flexible, centralized, and convenient way to go if you're managing a bunch of machines. Pull mode is something you can set up on any Windows Server 2012 R2 computer, and it doesn't even need to belong to a domain. If you're using the usual web server style of pull server (as opposed to SMB), you can configure either HTTP or HTTPS at your leisure (HTTPS merely requires an SSL certificate on the server).

In this guide, we're going to go through pretty much every aspect of DSC. The things we configure will be simple, so that we're not distracting from the discussion on DSC itself. This guide will evolve over time; if you notice blank sections, it's because those haven't yet been written. Errors, requests for more information, and so on should be reported in the PowerShell Q&A forum at PowerShell.org.

www.dbooks.org

# 4 Why MOF?

You'll notice that DSC has a heavy dependency on MOF files, and there's a good reason for it.

The Managed Object Format (MOF) was defined by the Distributed Management Task Force (DMTF), a vendor-neutral industry organization that Microsoft belongs to. The purpose of the DMTF is to supervise standards that help enable cross-platform management. In other words, MOF is a cross-platform standard. That means a couple of important things:

- You don't necessarily have to write a PowerShell configuration script to produce a MOF file that DSC can use. So long as you give it a valid MOF file, DSC is happy, no matter who produced that MOF file. This opens up the possibility of using third-party management tools.
- Because a PowerShell configuration script produces standard MOF file, you can potentially write configuration scripts that manage non-Windows computers. Remember, your PowerShell script doesn't get sent to target nodes. The script produces a MOF, which is sent to target nodes. If the target node is a Linux computer that knows what to do with a MOF file, then you're good to go.

So the idea behind MOFs is to create a configuration management system that's cross-platform compatible. Existing configuration management systems in the Linux world (think Chef and Puppet, for example) already use MOFs the same way. So Microsoft isn't tying you to using their technology: you can manage Windows servers using anything that's capable of producing a valid MOF.

MOFs are also closely related to the Common Information Model (CIM), another DMTF standard that Microsoft originally implemented as Windows Management Instrumentation (WMI). The MOF format is used to define the classes that appear in the CIM repository - another Microsoft attempt to work and play well with others, since non-Windows computers can also implement a CIM repository that's cross-platform compatible.

It's important to understand that DSC consists of three layers:

- Top: The domain-specific language (DSL) that you use to write declarative configuration scripts. This is a PowerShell language subset, and PowerShell compiles the scripts into MOFs.
- Middle: The node-side functionality that accepts MOFs and accomplishes the configuration process.
- Bottom: The node-side DSC resources that are called on by the MOFs to actually perform configuration tasks.

The point of this is that you can swap out the top layer for *anything* capable of producing the right MOF - and that the MOF format is vendor-neutral and not Microsoft-proprietary. Since anyone can provide elements of the bottom layer (and Microsoft will provide a lot), *any* management tool can leverage DSC. So if you've got a cross-platform management tool, and it can produce the right MOFs, then you don't necessarily need that tool's agent software installed on your computers. Instead, the tool can produce MOFs that tell the Windows DSC components what to do.

www.dbooks.org

# 5 Where to find Resources

WMF 4.0 ships with some basic DSC resources. As time goes on, it's likely that many Microsoft server products will include their own DSC resources for configuring those products. Also, Microsoft is releasing "out of band" waves of additional resources via the web.

- Wave 1: http://blogs.msdn.com/b/powershell/archive/2013/12/26/holiday-gift-desired-state-configuration-dsc aspx
- Wave 2: http://blogs.msdn.com/b/powershell/archive/2014/02/07/need-more-dsc-resources-announcing-dsc-r aspx?utm\char"005C\relax{}_source=tuicool

You'll find some open-source community resources at http://github.com/powershellorg/dsc, and a Google or Bing search will likely turn up other folks' DSC projects.

A word on naming: the Microsoft "waves" of resources all have resource names that start with the letter "x," which stands for "experimental, unsupported, use at your own risk." Keep in mind that these are *just PowerShell script modules*, which means they're basically open-source. Microsoft has asked that any community derivatives or original work carry the letter "c" as a resource name prefix. The idea is that non-prefixed names (e.g., "WindowsFeature") is reserved for Microsoft. By not using non-prefixed names, you will avoid coming up with a resource that conflicts with later Microsoft releases. Within your own organization, adopt a resource name prefix that's unique to you, like "contosoBusinessApp" for a "BusinessApp" resource owned by Contoso.

## Microsoft-Provided Resources

Microsoft provides the following resources in WMF 4:

- Registry
- Script (runs custom scripts; kind of a catch-all when there isn't a resource to do what you want)
- Archive (zip and unzip files)
- File
- WindowsFeature
- Package (install MSI or Setup.exe)
- Environment