wINDOWS MANAGEMEnt Framework 5.0

Preview September 2014   
Release Notes

# Overview

Windows Management Framework (WMF) 5.0 Preview September 2014 contains functionality that has been updated from WMF 4.0. WMF 5.0 Preview September 2014 is available for installation on Windows 8.1 and Windows Server 2012 R2, and contains updated versions of the following features:

* Windows PowerShell
* Windows PowerShell Integrated Scripting Environment (ISE)
* Windows PowerShell Web Services (Management OData IIS Extension)
* Windows PowerShell Desired State Configuration (DSC)
* Windows Remote Management (WinRM)
* Windows Management Instrumentation (WMI)

# Requirements

WMF 5.0 Preview September 2014 can be installed only on the following operating systems:

|  |  |  |
| --- | --- | --- |
| Operating System | Service Pack Level | Editions |
| Windows 8.1 |  | All |
| Windows Server 2012 R2 |  | All except IA64 |

Earlier releases of WMF 5.0 Preview (such as the May 2014 release) are not required to install WMF 5.0 Preview September 2014.

Install the latest Windows updates before installing WMF 5.0 Preview September 2014.

# How to install and uninstall WMF 5.0 Preview September 2014

## Installation instructions

### Windows 8.1 and Windows Server 2012 R2

Ensure all of the following prerequisites have been met:

* **Windows 8.1** or **Windows Server 2012 R2**
* **Windows PowerShell 2.0 (if applicable)**
* If you installed WMF 5.0 Experimental Release July 2014, you must uninstall it before installing WMF 5.0 Preview September 2014.
* Windows PowerShell 2.0 is disabled by default on Windows Server 2012 R2 Server Core. It can be enabled by running the following two commands:

dism /online /enable-feature:MicrosoftWindowsPowerShellV2

dism /online /enable-feature:MicrosoftWindowsPowerShellV2-WOW64

Installation:

* **Double-click** the MSU file to start installation, or run the MSU file directly from **Command Prompt**.

## Uninstallation instructions

### By using Control Panel

1. Open **Control Panel**.
2. Open **Programs**, then open **Uninstall a program**.
3. Click **View installed updates**.
4. Select **Windows Management Framework 5.0 Preview September 2014** from the list of installed updates. This corresponds to KB2969050. Click **Uninstall.**

### By using Command Prompt

1. Open **Command Prompt**.
2. Run the following command:

wusa /uninstall /kb: KB2969050

# New scenarios enabled by WMF 5.0 Preview September 2014

## Generate Windows PowerShell cmdlets based on an OData endpoint

Export-ODataEndpointProxy is a cmdlet that generates a set of Windows PowerShell cmdlets based on the functionality exposed by a given OData Endpoint.

The following example shows how to use this new cmdlet:

# Basic use case of Export-ODataEndpoint Proxy

Export-ODataEndpointProxy -Uri 'http://services.odata.org/v3/(S(snyobsk1hhutkb2yulwldgf1))/odata/odata.svc' -OutputPath C:\Users\user\Generated.psd1

ipmo 'C:\Users\user\Generated.psd1'

# Cmdlets are created based on the following heuristics

# New-<EntityType> -<Key> [-<Other Attributes>…]

#

# Get-<EntityType> [-<Key> -Top –Skip –Filter -OrderBy]

# # If there is a complex key, the keys will actually be -<Key1> -<Key2>…

# # Note this rule applies to any other instances of the key

#

# Set-<EntityType> -<Key> [-<Other Attributes>]

#

# Remove-<EntityType> -<Key>

#

# Invoke-<EntityType><Action> [-<Key> -<Other Parameters>]

#

#

# Cmdlets from associations (Note: Get and Remove get additional parameter sets)

# Get-<EntityType> -<AssociatedEntity>

# New-<EntityType> -<AssociatedEntity> -<Key>

# Remove-<EntityType> -<AssociatedEntity> -<Key>

#

#

# Note: Every cmdlet has a –ConnectionURI parameter for explicitly setting the URI of the endpoint. This normally uses the same address that you gave the Export-ODataEndpointProxy cmdlet, but can be overridden in this fashion for the sake of similar endpoints.

#

There are still parts of key use cases in development for this functionality, including, but not limited to:

* Associations
* Passing streams

## Manage .ZIP files with new cmdlets

Two new cmdlets, Compress-Archive and Expand-Archive, let you compress and expand ZIP files.

### Compress-Archive

The Compress-Archive cmdlet creates a new archive file from specified files. An archive file allows multiple files to be packaged and optionally compressed into a single file for easier handling and storage. An archive file can be compressed by using a compression algorithm specified in the -CompressionLevel parameter.

|  |
| --- |
| Compress-Archive -LiteralPath <String[]> [-DestinationPath] <String> [-Update] [-CompressionLevel <Microsoft.PowerShell.Commands.CompressionLevel>] |
| Compress-Archive [-Path] <String[]> [-DestinationPath] <String> [-Update] [-CompressionLevel <Microsoft.PowerShell.Commands.CompressionLevel>] |

### Expand-Archive

The Expand-Archive cmdlet extracts files from a specified archive file. An archive file allows multiple files to be packaged and optionally compressed into a single file for easier handling and storage.

|  |
| --- |
| Expand-Archive -LiteralPath <String> [-DestinationPath] <String> |
| Expand-Archive [-Path] <String> [-DestinationPath] <String> |

## Support your configuration environment with new features and cmdlets

Features and cmdlets have been added to Windows PowerShell Desired State Configuration (DSC) to enable you to better understand, test, and manage your configurations.

### Authoring improvements in Windows PowerShell ISE

Authoring DSC configurations in Windows PowerShell ISE is much easier, thanks to the following improvements:

* List all DSC resources within a **configuration** block or **node** block by entering **Ctrl+Space** on a blank line within it.
* Automatic completion on resource properties that are of the **enumeration** type.
* Automatic completion on the **DependsOn** property of DSC resources, based on other resource instances in the configuration.
* Better tab completion of resource property values.

**Note:** You must have an empty string for resource property values before you can use Ctrl+Space to list the options. Pressing **Tab** cycles through options.

### New attribute for defining meta-configuration

The **DscLocalConfigurationManager** attribute designates a configuration block as a meta-configuration, which is used to configure the DSC Local Configuration Manager. This attribute restricts a configuration to containing only items which configure the DSC Local Configuration Manager. During processing, this configuration generates a \*.meta.mof file that is then sent to the appropriate target nodes by using the Set-DscLocalConfigurationManager cmdlet.

[DscLocalConfigurationManager()]

configuration meta

{

Node localhost

{

LocalConfigurationManager

{

ConfigurationMode = "Pull"

ConfigurationID = "5603f952-d6c6-4971-88c4-948636bf5c3f"

RefreshMode = "ApplyAndAutocorrect"

}

ConfigurationRepositoryWeb PullServer

{

ServerURL = "https://corp.contoso.com/PSDSCPullServer/PSDSCPullServer.svc"

}

}

}

The above example configures the LCM for pull mode, changes the refresh mode to **Autocorrect**, and defines the type and location of the pull servers.

This new configuration replaces and extends the functionality of the LocalConfigurationManager resource from DSC v1. LocalConfigurationManager is still supported in configurations without this attribute, for backwards compatibility.

Meta-resources:

|  |  |
| --- | --- |
| Resource Name | Description |
| LocalConfigurationManager | Various settings for DSC engine execution |
| PartialConfiguration | Partial configuration settings |
| ConfigurationRepositoryWeb | Web-based configuration repository |
| ConfigurationRepositoryShare | File share-based configuration repository |
| MSFT\_WebResourceManager | Web-based resource repository |
| MSFT\_FileResourceManager | File-based resource repository |
| MSFT\_WebReportManager | Web-based reporting endpoint for pull scenario |

### Partial configurations

This release of WMF helps you deliver configuration documents to a node in fragments. For a node to receive multiple fragments of a configuration document, its Local Configuration Manager must be set first to specify the expected fragments, such as:

[DSCLocalConfigurationManager()]

configuration SQLServerDSCSettings

{

Node localhost

{

LocalConfigurationManager

{

ConfigurationModeFrequencyMins = 30

}

ConfigurationRepositoryWeb OSConfigServer

{

ServerURL = "https://corp.contoso.com/OSConfigServer/PSDSCPullServer.svc"

}

ConfigurationRepositoryWeb SQLConfigServer

{

ServerURL = "https://corp.contoso.com/SQLConfigServer/PSDSCPullServer.svc"

}

PartialConfiguration OSConfig

{

Description = 'Configuration for the Base OS'

ExclusiveResources = 'PSDesiredStateConfiguration\\*'

ConfigurationSource = '[ConfigurationRepositoryWeb]OSConfigServer'

}

PartialConfiguration SQLConfig

{

Description = 'Configuration for the SQL Server'

ConfigurationSource = '[ConfigurationRepositoryWeb]SQLConfigServer'

DependsOn = '[PartialConfiguration]OSConfig'

}

}

}

While authoring the partial configuration, the configuration name must match what is defined in the Local Configuration Manager. In the above example, the configurations should be named **OSConfig** and **SQLConfig**.

**NOTE: In this release, this feature works in Pull mode only.** Setting Local Configuration Manager for Partial Configuration enables configuration coordination, but does NOT provide security functionality.

### Cross-computer synchronization

By using the built-in WaitFor\* resources (WaitForAll, WaitForAny, and WaitForSome), you can now specify dependencies across computers during configuration runs, without external orchestrations. These resources provide node-to-node synchronization by using CIM connections over the WS-Man protocol. By using these resources, a configuration can wait for another computer’s specific resource state to change.

For example, in the following configuration, the target node is waiting for the **xADDomain** resource to finish on the **MyDC** node with a few retries, before the target node can join the domain.

configuration JoinDomain

{

Import-DscResource -Module xComputerManagement

WaitForAll DC

{

ResourceName = '[xADDomain]NewDomain'

NodeName = 'MyDC'

RetryIntervalSec = 15

RetryCount = 30

}

xComputer JoinDomain

{

Name = 'MyPC'

DomainName = 'Contoso.com'

Credential = (get-credential)

DependsOn ='[WaitForAll]DC'

}

}

### Get-DscConfigurationStatus

The Get-DscConfigurationStatus cmdlet gets high level information about configuration status from a target node. You can obtain the status of the last or all configurations.

Get-DscConfigurationStatus -All [-CimSession <CimSession[]>] [-ThrottleLimit <int>] [-AsJob] [<CommonParameters>]

Get-DscConfigurationStatus [-CimSession <CimSession[]>] [-ThrottleLimit <int>] [-AsJob] [<CommonParameters>]

### Compare-DscConfiguration

The Compare-DscConfiguration cmdlet compares a specified configuration with the actual state of one or more target nodes.

Compare-DscConfiguration [[-ReferenceConfiguration] <string>] [[-ComputerName] <string[]>] [-AsJob] [-Credential <pscredential>] [-ThrottleLimit <int>] [<CommonParameters>]

Compare-DscConfiguration [[-ReferenceConfiguration] <string>] -CimSession <CimSession[]> [-AsJob] [-throttleLimit <int>] [<CommonParameters>]

### Publish-DscConfiguration

The Publish-DscConfiguration cmdlet copies a configuration MOF file to a target node, but does not apply the configuration. This configuration is applied during the next consistency pass, or when you run the Update-DscConfiguration cmdlet.

Publish-DscConfiguration [-Path] <string> [[-ComputerName] <string[]>] [-Force] [-Credential <pscredential>]

[-ThrottleLimit <int>] [-WhatIf] [-Confirm] [<CommonParameters>]

Publish-DscConfiguration [-Path] <string> -CimSession <CimSession[]> [-Force] [-ThrottleLimit <int>] [-WhatIf]

[-Confirm] [<CommonParameters>]

### Update-DscConfiguration

The Update-DscConfiguration cmdlet forces the configuration to be processed. If the LCM is in pull mode, it gets the configuration from the pull server before applying it.

Update-DscConfiguration [[-ComputerName] <string[]>] [-Wait] [-Force] [-JobName <string>] [-Credential

<pscredential>] [-ThrottleLimit <int>] [-WhatIf] [-Confirm] [<CommonParameters>]

Update-DscConfiguration -CimSession <CimSession[]> [-Wait] [-Force] [-JobName <string>] [-ThrottleLimit <int>]

[-WhatIf] [-Confirm] [<CommonParameters>]

## Audit Windows PowerShell usage by transcription and logging

### Enhanced Windows PowerShell Transcription options

Windows PowerShell transcription has been improved to apply to all hosting applications (such as PowerShell ISE) rather than just the console host (powershell.exe).

In addition to extending for transcripting, the transcripting functionality itself has been updated to support arbitrary nesting of transcripts, additional metadata in the resulting transcript header, and the ability to set a transcription output directory (to support centralized log collection).

Transcription options (including enabling a system-wide transcript) can be configured by configuring the **Turn on PowerShell Transcription** Group Policy setting (in Administrative Templates -> Windows Components -> Windows PowerShell).

### Script tracing and logging

While Windows PowerShell already has the **LogPipelineExecutionDetails** Group Policy setting to log the invocation of cmdlets, Windows PowerShell’s scripting language has plenty of features that you might want to log and/or audit. The new Detailed Script Tracing feature lets you enable detailed tracking and analysis of Windows PowerShell scripting use on a system. After you enable detailed script tracing, Windows PowerShell logs all script blocks to the ETW event log, **Microsoft-Windows-PowerShell/Operational**. If a script block creates another script block (for example, a script that calls the Invoke-Expression cmdlet on a string), that resulting script block is logged as well.

Logging of these events can be enabled through the **Turn on PowerShell Script Block Logging** Group Policy setting (in Administrative Templates -> Windows Components -> Windows PowerShell).

The events are:

|  |  |
| --- | --- |
| Channel | Operational |
| Level | Verbose |
| Opcode | Create |
| Task | CommandStart |
| Keyword | Runspace |
| EventId | Engine\_ScriptBlockCompiled (0x1008 = 4104) |
| Message | Creating Scriptblock text (%1 of %2):  %3  ScriptBlock ID: %4 |

The text embedded in the message is the extent of the script block compiled. The ID is a GUID that is retained for the life of the script block.

When you enable verbose logging, the feature writes begin and end markers:

|  |  |
| --- | --- |
| Channel | Operational |
| Level | Verbose |
| Opcode | Open (/ Close) |
| Task | CommandStart (/ CommandStop) |
| Keyword | Runspace |
| EventId | ScriptBlock\_Invoke\_Start\_Detail (0x1009 = 4105) /  ScriptBlock\_Invoke\_Complete\_Detail (0x100A = 4106) |
| Message | Started (/ Completed) invocation of ScriptBlock ID: %1  Runspace ID: %2 |

The ID is the GUID representing the script block (that can be correlated with event ID 0x1008), and the Runspace ID represents the runspace in which this script block was run.

Percent signs in the invocation message represent structured ETW properties. While they are replaced with the actual values in the message text, a more robust way to access them is to retrieve the message with the Get-WinEvent cmdlet, and then use the **Properties** array of the message.

Here's an example of how this functionality can help unwrap a malicious attempt to encrypt and obfuscate a script:

## Malware

function SuperDecrypt

{

    param($script)

    $bytes = [Convert]::FromBase64String($script)

    ## XOR “encryption”

    $xorKey = 0x42

    for($counter = 0; $counter -lt $bytes.Length; $counter++)

    {

        $bytes[$counter] = $bytes[$counter] -bxor $xorKey

    }

    [System.Text.Encoding]::Unicode.GetString($bytes)

}

$decrypted = SuperDecrypt "FUIwQitCNkInQm9CCkItQjFCNkJiQmVCEkI1QixCJkJlQg=="

Invoke-Expression $decrypted

Running this generates the following log entries:

Compiling Scriptblock text (1 of 1):

function SuperDecrypt

{

param($script)

$bytes = [Convert]::FromBase64String($script)

## XOR "encryption"

$xorKey = 0x42

for($counter = 0; $counter -lt $bytes.Length; $counter++)

{

$bytes[$counter] = $bytes[$counter] -bxor $xorKey

}

[System.Text.Encoding]::Unicode.GetString($bytes)

}

ScriptBlock ID: ad8ae740-1f33-42aa-8dfc-1314411877e3

Compiling Scriptblock text (1 of 1):

$decrypted = SuperDecrypt "FUIwQitCNkInQm9CCkItQjFCNkJiQmVCEkI1QixCJkJlQg=="

ScriptBlock ID: ba11c155-d34c-4004-88e3-6502ecb50f52

Compiling Scriptblock text (1 of 1):

Invoke-Expression $decrypted

ScriptBlock ID: 856c01ca-85d7-4989-b47f-e6a09ee4eeb3

Compiling Scriptblock text (1 of 1):

Write-Host 'Pwnd'

ScriptBlock ID: 5e618414-4e77-48e3-8f65-9a863f54b4c8

If the script block length exceeds what ETW is capable of holding in a single event, Windows PowerShell breaks the script into multiple parts. Here is sample code to recombine a script from its log messages:

    $created = Get-WinEvent -FilterHashtable @{ ProviderName="Microsoft-Windows-PowerShell"; Id = 4104 } |

        Where-Object { $\_.<...> }

    $sortedScripts = $created | sort { $\_.Properties[0].Value }

    $mergedScript = -join ($sortedScripts | % { $\_.Properties[2].Value })

As with all logging systems that have a limited retention buffer (i.e.: ETW logs), one attack against this infrastructure is to flood the log with spurious events to hide earlier evidence. To protect yourself from this attack, ensure that you have some form of event log collection set up (i.e.: Windows Event Forwarding, <http://www.nsa.gov/ia/_files/app/Spotting_the_Adversary_with_Windows_Event_Log_Monitoring.pdf>) to move event logs off of the computer as soon as possible.

### CMS (Cryptographic Message Syntax) Cmdlets

The Cryptographic Message Syntax cmdlets support encryption and decryption of content using the IETF standard format for cryptographically protecting messages as documented by [RFC5652](http://tools.ietf.org/html/rfc5652).

  Get-CmsMessage [-Content] <string>

  Get-CmsMessage [-Path] <string>

  Get-CmsMessage [-LiteralPath] <string>

  Protect-CmsMessage [-To] <CmsMessageRecipient[]> [-Content] <string> [[-OutFile] <string>]

  Protect-CmsMessage [-To] <CmsMessageRecipient[]> [-Path] <string> [[-OutFile] <string>]

  Protect-CmsMessage [-To] <CmsMessageRecipient[]> [-LiteralPath] <string> [[-OutFile] <string>]

  Unprotect-CmsMessage [-EventLogRecord] <EventLogRecord> [[-To] <CmsMessageRecipient[]>] [-IncludeContext]

  Unprotect-CmsMessage [-Content] <string> [[-To] <CmsMessageRecipient[]>] [-IncludeContext]

  Unprotect-CmsMessage [-Path] <string> [[-To] <CmsMessageRecipient[]>] [-IncludeContext]

  Unprotect-CmsMessage [-LiteralPath] <string> [[-To] <CmsMessageRecipient[]>] [-IncludeContext]

The CMS encryption standard implements public key cryptography, where the keys used to encrypt content (the *public key*) and the keys used to decrypt content (the *private key*) are separate.

Your public key can be shared widely, and is not sensitive data. If any content is encrypted with this public key, only your private key can decrypt it. For more information about Public Key Cryptography, see: <http://en.wikipedia.org/wiki/Public-key_cryptography>.

To be recognized in Windows PowerShell, encryption certificates require a unique key usage identifier (EKU) to identify them as data encryption certificates (like the identifiers for 'Code Signing', 'Encrypted Mail').

Here is an example of creating a certificate that is good for Document Encryption:

(Change the text in **Subject** to your name, email, or other identifier), and put in a file (i.e.: DocumentEncryption.inf):

  [Version]

  Signature = "$Windows NT$"

  [Strings]

  szOID\_ENHANCED\_KEY\_USAGE = "2.5.29.37"

  szOID\_DOCUMENT\_ENCRYPTION = "1.3.6.1.4.1.311.80.1"

  [NewRequest]

  Subject = "[cn=me@somewhere.com](mailto:cn=me@somewhere.com)"

  MachineKeySet = false

  KeyLength = 2048

  KeySpec = AT\_KEYEXCHANGE

  HashAlgorithm = Sha1

  Exportable = true

  RequestType = Cert

  KeyUsage = "CERT\_KEY\_ENCIPHERMENT\_KEY\_USAGE | CERT\_DATA\_ENCIPHERMENT\_KEY\_USAGE"

  ValidityPeriod = "Years"

  ValidityPeriodUnits = "1000"

  [Extensions]

  %szOID\_ENHANCED\_KEY\_USAGE% = "{text}%szOID\_DOCUMENT\_ENCRYPTION%"

Then run:

  certreq -new DocumentEncryption.inf DocumentEncryption.cer

And you can now encrypt and decrypt content:

106 [C:\temp]  
>> $protected = "Hello World" | Protect-CmsMessage -To "\*me@somewhere.com\*"

107 [C:\temp]  
>> $protected  
  
-----BEGIN CMS-----  
MIIBqAYJKoZIhvcNAQcDoIIBmTCCAZUCAQAxggFQMIIBTAIBADA0MCAxHjAcBgNVBAMMFWxlZWhv  
bG1AbWljcm9zb2Z0LmNvbQIQQYHsbcXnjIJCtH+OhGmc1DANBgkqhkiG9w0BAQcwAASCAQAnkFHM  
proJnFy4geFGfyNmxH3yeoPvwEYzdnsoVqqDPAd8D3wao77z7OhJEXwz9GeFLnxD6djKV/tF4PxR  
E27aduKSLbnxfpf/sepZ4fUkuGibnwWFrxGE3B1G26MCenHWjYQiqv+Nq32Gc97qEAERrhLv6S4R  
G+2dJEnesW8A+z9QPo+DwYU5FzD0Td0ExrkswVckpLNR6j17Yaags3ltNVmbdEXekhi6Psf2MLMP  
TSO79lv2L0KeXFGuPOrdzPAwCkV0vNEqTEBeDnZGrjv/5766bM3GW34FXApod9u+VSFpBnqVOCBA  
DVDraA6k+xwBt66cV84OHLkh0kT02SIHMDwGCSqGSIb3DQEHATAdBglghkgBZQMEASoEEJbJaiRl  
KMnBoD1dkb/FzSWAEBaL8xkFwCu0e1ZtDj7nSJc=  
-----END CMS-----

108 [C:\temp]  
>> $protected | Unprotect-CmsMessage  
Hello World

Any parameter of type CMSMessageRecipient supports identifiers in the following formats:

* An actual certificate (as retrieved from the certificate provider)
* Path to the a file containing the certificate
* Path to a directory containing the certificate
* Thumbprint of the certificate (used to look in the certificate store)
* Subject name of the certificate (used to look in the certificate store)

To view document encryption certificates in the certificate provider, you can use the -**DocumentEncryptionCert** dynamic parameter:

58 [Cert:\currentuser\my]

>> dir -DocumentEncryptionCert

## Extract and parse structured objects out of string content

In collaboration with Microsoft Research, a new **ConvertFrom-String** cmdlet has been added.

This cmdlet supports two modes: basic delimited parsing, and auto generated example-driven parsing.

Delimited parsing, by default, splits the input at white space, and assigns property names to the resulting groups. You can customize the delimiter:

1 [C:\temp]  
>> "Hello World" | ConvertFrom-String | Format-Table -Auto

P1    P2  
--    --The cmdlet also supports auto-generated example-driven parsing based on the FlashExtract research work in Microsoft Research.

To get started, consider a text-based address book:

    Ana Trujillo

    Redmond, WA

    Antonio Moreno

    Renton, WA

    Thomas Hardy

    Seattle, WA

    Christina Berglund

    Redmond, WA

    Hanna Moos

    Puyallup, WA

Copy a few examples into a file, which you will use as your template:

    Ana Trujillo

    Redmond, WA

    Antonio Moreno

    Renton, WA

Put curly braces around data that you want to extract, giving it a name as you do so. Since the **Name** property (and its associated otherproperties) can appear multiple times, use an asterisk (\*) to indicate that this results in multiple records (rather than extracting a bunch of properties into one record):

    {Name\*:Ana Trujillo}

    {City:Redmond}, {State:WA}

    {Name\*:Antonio Moreno}

    {City:Renton}, {State:WA}

From this set of examples, ConvertFrom-String can now automatically extract object-based output from input files with similar structure.

2 [C:\temp]

>> Get-Content .\addresses.output.txt | ConvertFrom-String -TemplateFile .\addresses.template.txt |  
>>> Format-Table -Auto

ExtentText                     Name               City     State  
----------                     ----               ----     -----  
Ana Trujillo...                Ana Trujillo       Redmond  WA  
Antonio Moreno...              Antonio Moreno     Renton   WA  
Thomas Hardy...                Thomas Hardy       Seattle  WA  
Christina Berglund...          Christina Berglund Redmond  WA  
Hanna Moos...                  Hanna Moos         Puyallup WA

To do additional data manipulation on extracted text, the **ExtentText** property captures the raw text from which the record was extracted. To provide feedback on this feature, or to share content that you are having difficulty writing examples for, please email [psdmfb@microsoft.com](mailto:psdmfb@microsoft.com).

## Extending the Item noun to enable symbolic links

To support symbolic links, \*-Item and a few related cmdlets have been extended. Now you can create symbolic links in a single, simple line with New-Item.

You’ll notice that the Item related cmdlets (Remove-Item, Get-ChildItem) behave very similarly to before.

The following shows some use cases of the new capabilities:

### SYMBOLIC LINK FILES

# Create a new symbolic link file named MySymLinkFile.txt in C:\Temp which links to $pshome\profile.ps1

cd C:\Temp

New-Item -ItemType SymbolicLink -Name MySymLinkFile.txt -Target $pshome\profile.ps1 # File

# Target is an alias to the Value parameter

# Equivalent to above

New-Item -ItemType SymbolicLink -Path C:\Temp -Name MySymLinkFile.txt -Value $pshome\profile.ps1

# Equivalent to above

New-Item -ItemType SymbolicLink -Path C:\Temp\MySymLinkFile.txt -Value $pshome\profile.ps1

# Equivalent to above

New-Item -ItemType SymbolicLink -Name C:\Temp\MySymLinkFile.txt -Value $pshome\profile.ps1

### SYMBOLIC LINK DIRECTORIES

# Create a new symbolic link directory named MySymLinkDir in C:\Temp which links to the $pshome folder

# ItemType is the same for files and directories - autodetect based on specified target

cd C:\Temp

New-Item -ItemType SymbolicLink -Name MySymLinkDir -Target $pshome # Directory

# Target is an alias to the Value parameter

# Similar to above, any combination of Path and Name also works

New-Item -ItemType SymbolicLink -Path C:\Temp -Name MySymLinkDir -Value $pshome

New-Item -ItemType SymbolicLink -Path C:\Temp\MySymLinkDir -Value $pshome

New-Item -ItemType SymbolicLink -Name C:\Temp\MySymLinkDir -Value $pshome

### HARD LINKS

New-Item -ItemType HardLink -Path C:\Temp -Name MyHardLinkFile.txt -Value $pshome\profile.ps1

# Same combinations of Path and Name allowed as described above

### DIRECTORY JUNCTIONS

New-Item -ItemType Junction -Path C:\Temp\MyJunctionDir -Value $pshome

# Same combinations of Path and Name allowed as described above

# GET-CHILDITEM

# Append link type column to Mode property and display with Get-ChildItem

# Use 'l' for all link types

# Increase the width of the Length column by 4 (from 10 to 14)

Get-ChildItem C:\Temp | sort LastWriteTime -Descending

Directory: C:\Temp

Mode LastWriteTime Length Name

---- ------------- ------ ----

-a---- 6/13/2014 3:00 PM 16 File.txt

-a---- 6/13/2014 3:00 PM 98956046499840 My90TB.vhd

d----- 6/13/2014 3:00 PM Directory

-a---l 6/13/2014 3:21 PM 0 MySymLinkFile.txt

d----l 6/13/2014 3:22 PM MySymLinkDir

-a---l 6/13/2014 3:23 PM 23304 MyHardLinkFile.txt

d----l 6/13/2014 3:24 PM MyJunctionDir

# New Target property

# Works with any link type

# Not displayed in the default table view

# Displayed in the default list view

# New LinkType property with values: SymbolicLink

Get-ChildItem C:\Temp\MySymLinkFile.txt | Format-List

Directory: C:\Temp

Name : MySymLinkFile.txt

Length : 0

Mode : -a---l

LinkType : SymbolicLink

Target : C:\Windows\System32\WindowsPowerShell\v1.0\profile.ps1

CreationTime : 6/16/2014 3:21:01 PM

LastWriteTime : 6/16/2014 3:21:01 PM

LastAccessTime : 6/16/2014 3:21:01 PM

VersionInfo : File: C:\Temp\MySymLinkFile.txt

InternalName:

OriginalFilename:

FileVersion:

FileDescription:

Product:

ProductVersion:

Debug: False

Patched: False

PreRelease: False

PrivateBuild: False

SpecialBuild: False

Language:

# REMOVE-ITEM

# Works like any other item type

# Removes MySymLinkFile.txt

Remove-Item C:\Temp\MySymLinkFile.txt

# Returns an error as this is a reparse point.

Remove-Item C:\Temp\MySymLinkDir

# Removes the files under the target directory and MySymLinkDir

Remove-Item C:\Temp\MySymLinkDir -Force

## Develop with classes in Windows PowerShell

WMF 5.0 Preview September 2014 adds support to the Windows PowerShell language for defining classes and other user-defined types, by using formal syntax and semantics that are similar to other object-oriented programming languages. The goal is to enable developers and IT professionals to embrace Windows PowerShell for a wider range of use cases, simplify development of Windows PowerShell artifacts (such as DSC resources), and accelerate coverage of management surfaces.

### Supported scenarios in this release

* Define DSC resources and their associated types by using the Windows PowerShell language.
* Define custom types in Windows PowerShell by using familiar object-oriented programming constructs, such as classes, properties, methods, inheritance, etc.
* Debug types by using the Windows PowerShell language.
* Generate and handle exceptions by using formal mechanisms, and at the right level.

### Define DSC resources with classes

In this release of WMF 5.0, the Windows PowerShell language enhancements that let you define DSC resources by using classes are the most complete. Apart from syntax changes, the major differences between a class-defined DSC resource and a cmdlet DSC resource provider are the following.

* A MOF file is not required.
* A **DSCResource** subfolder in the module folder is not required.
* A Windows PowerShell module file can contain multiple DSC resource classes.

The following is an example of a class-defined DSC resource provider; this is saved as a module, **MyDSCResource.psm1**. Note that you must always include a key property in a class-defined DSC resource provider.

# This module defines a class for a DSC "FileResource" provider.

enum Ensure

{

Absent

Present

}

<# This resource manages the file in a specific path.

[DscResource()] indicates the class is a DSC resource

#>

[DscResource()]

class FileResource

{

<# This is a key property

[DscResourceKey()] also means the property is required.

#>

[DscResourceKey()][string]$Path

[Ensure] $Ensure

<#

[DscResourceMandatory()] means the property is required.

#>

[DscResourceMandatory()]

[string] $SourcePath

#>

<#

This method replaces the Set-TargetResource DSC script function.

It sets the resource to the desired state. The DSC engine sets

all of the properties from the configuration.

[DscResourceKey()] or [DscResourceMandatory()] are guaranteed to

be set, other properties may not be set if the configuration did

not specify values.

#>

[void] Set()

{

if($ensure -eq [Ensure]::Present)

{

if (!(Test-Path $Path))

{

$err = Copy-Item $SourcePath $Path

if ($err)

{

Write-Verbose ("(ERROR) Failed to copy file from '{0}' to '{1}'", $SourcePath, $Path)

throw $err

}

}

}

else

{

if (Test-Path $Path)

{

del $Path

}

}

}

<#

This method replaces the Test-TargetResource function.

It should return True or False, showing whether the resource is in a desired state.

The DSC engine sets all of the properties from the configuration.

Properties with [DscResourceKey()] or [DscResourceMandatory()] are

guaranteed to be set, other properties may not be set if the

configuration did not specify values.

#>

[bool] Test()

{

if($ensure -eq [Ensure]::Present)

{

return Test-Path $Path

}

else

{

return !(Test-Path $Path)

}

}

<#

This method replaces the Get-TargetResource function.

The DSC engine sets all of the properties from the configuration.

Properties with [DscResourceKey()] or [DscResourceMandatory()] are

guaranteed to be set; other properties may not be set if the

configuration did not specify values.

The implementation should use the keys to find appropriate resources.

This method returns an instance of this class with the updated key properties.

#>

[FileResource] Get()

{

$file = Get-item $Path

return $this

}

After creating the class-defined DSC resource provider, and saving it as a module, create a module manifest for the module. In this example, the following module manifest is saved as **MyDscResource.psd1**.

@{

# Script module or binary module file associated with this manifest.

RootModule = 'MyDscResource.psm1'

# Version number of this module.

ModuleVersion = '1.0'

# ID used to uniquely identify this module

GUID = '*GUID*'

# Author of this module

Author = '*Author name*'

# Company or vendor of this module

CompanyName = 'Microsoft

# Copyright statement for this module

Copyright = '(c) 2014 Microsoft. All rights reserved.'

# Description of the functionality provided by this module

# Description = 'DSC resource provider for FileResource.'

# Minimum version of the Windows PowerShell engine required by this module

# PowerShellVersion = ''

# Name of the Windows PowerShell host required by this module

# PowerShellHostName = ''

}

Deploy the new DSC resource provider by creating a **MyDscResource** folder for it in $pshome\Modules or $env:SystemDrive\Program Files\WindowsPowerShell\Modules. You do not need to create a DSCResource subfolder. Copy the module and module manifest files (**MyDscResource.psm1** and **MyDscResource.psd1**) to the **MyDscResource** folder.

From this point, you create and run a configuration script as you would with any DSC resource. The following is a configuration that references the MyDSCResource module. Save this as a script, **MyResource.ps1**.

Configuration Test

{

Import-DSCResource -module MyDscResource

FileResource file

{

Path = "C:\test\test.txt"

SourcePath = "c:\test.txt"

Ensure = "Present"

}

}

Test

Run this as you would any DSC configuration script. To start the configuration, in an elevated Windows PowerShell console, run the following.

PS C:\test> .\MyResource.ps1

### Define custom types in PowerShell

WMF 5.0 Preview September 2014 introduces the following new language elements in Windows PowerShell.

| Element | Description |
| --- | --- |
| **Class** keyword | Defines a new class. This is a true .NET Framework type.  Class members are public, but only public within the module scope. You can't refer to the type name as a string (for example, New-Object doesn't work), and in this release, you can't use a type literal (for example, [MyClass]) outside the script/module file in which the class is defined.  class MyClass  {  } |
| **Enum** keyword and enumerations | Support for the **enum** keyword has been added; this is a breaking change. The **enum** delimiter is currently a newline. A workaround for those who are already using **enum** is to insert an ampersand (&) before the word.  Current limitations: you cannot define an enumerator in terms of itself, but you can initialize enum in terms of another enum, as shown in the following example. The base type cannot currently be specified; it is always [int].  enum Color2  {  Yellow = [Color]::Blue  }  An enumerator value must be a parse time constant; you cannot set it to the result of an invoked command.  enum MyEnum  {  Enum1  Enum2  Enum3 = 42  Enum4 = [int]::MaxValue  }  Enums support arithmetic operations, as shown in the following example.  enum SomeEnum { Max = 42 }  enum OtherEnum { Max = [SomeEnum]::Max + 1 } |
| Import-DscResource | Import-DscResource is now a true dynamic keyword. Windows PowerShell parses the specified module’s root module, searching for classes that contain the **DscResource** attribute. |
| Properties | A new field, **ImplementingAssembly**, has been added to ModuleInfo. It is set to the dynamic assembly created for a script module if the script defines classes, or the loaded assembly for binary modules. It is not set when ModuleType = Manifest.  Reflection on the **ImplementingAssembly** field discovers resources in a module. This means you can discover resources written in either PowerShell or other managed languages.  Fields with initializers:  [int] $i = 5  Static is supported; it works like an attribute, as do the type constraints, so it can be specified in any order.  static [int] $count = 0  A type is optional.  $s = "hello"  All members are public. |
| Constructors and instantiation | Windows PowerShell classes can have constructors; they have the same name as their class. Constructors can be overloaded. Static constructors are supported. Properties with initialization expressions are initialized before running any code in a constructor. Static properties are initialized before the body of a static constructor, and instance properties are initialized before the body of the non-static constructor.Currently, there is no syntax for calling a constructor from another constructor (like the C# syntax ": this()"). The workaround is to define a common Init method.  The following are ways of instantiating classes in this release.  # Instantiating by using the default constructor. Note that New-Object is not supported in this release.  $a = [MyClass]::new()  # Calling a constructor with a parameter  $b = [MyClass]::new(42)  # Passing an array to a constructor with multiple parameters  $c = [MyClass]::new(@(42,43,44), "Hello")  In this release, New-Object does not work with classes defined in Windows PowerShell. Also for this release, the type name is only visible lexically, meaning it is not visible outside of the module or script that defines the class. Functions can return instances of a class defined in Windows PowerShell, and instances work well outside of the module or script.  Get-Member -Static lists constructors, so you can view overloads like any other method. The performance of this syntax is also considerably faster than New-Object.  The pseudo-static method named **new** works with .Net types, as shown in the following example.  [hashtable]::new()  You can now see constructor overloads with Get-Member, or as shown in this example:  PS> [hashtable]::new  OverloadDefinitions  -------------------  hashtable new()  hashtable new(int capacity)  hashtable new(int capacity, float loadFactor) |
| Methods | A Windows PowerShell class method is implemented as a ScriptBlock that has only an end block. All methods are public. The following shows an example of defining a method named **DoSomething**.  class MyClass  {  DoSomething($x)  {  $this.\_doSomething($x) # method syntax  }  private \_doSomething($a) {}  }  Method invocation:  $b = [MyClass]::new()  $b.DoSomething(42)  Overloaded methods—that is, those that are named the same as an existing method, but differentiated by their specified values—are also supported. |
| Properties | All properties are public. Properties require either a newline or semicolon. If not object type is specified, the property type is object. |
| Invocation | See “Method invocation” in this table. |
| Return types | Return type is a contract; the return value is converted to the expected type. If no return type is specified, the return type is void. There is no streaming of objects; objects cannot be written to the pipeline either intentionally or by accident. |
| Attributes | Three new attributes, **DscResource**, **DscResourceKey**, and **DscResourceMandatory**, have been added. |
| Lexical scoping of variables | The following shows an example of how lexical scoping works in this release.  $d = 42 # Script scope  function bar  {  $d = 0 # Function scope  [MyClass]::DoSomething()  }  class MyClass  {  static [object] DoSomething()  {  return $d # error, not found dynamically  return $script:d # no error  $d = $script:d  return $d # no error, found lexically  }  }  $v = bar  $v -eq $d # true |

The following example creates several new, custom classes to implement an HTML dynamic stylesheet language (DSL). Then, the example adds helper functions to create specific element types as part of the element class, such as heading styles and tables, because types cannot be used outside the scope of a module.

# Classes that define the structure of the document

#

class Html

{

[string] $docType

[HtmlHead] $Head

[Element[]] $Body

[string] Render()

{

$text = "<html>`n<head>`n"

$text += $Head

$text += "`n</head>`n<body>`n"

$text += $Body -join "`n" # Render all of the body elements

$text += "</body>`n</html>"

return $text

}

[string] ToString() { return $this.Render() }

}

class HtmlHead

{

$Title

$Base

$Link

$Style

$Meta

$Script

[string] Render() { return "<title>$Title</title>" }

[string] ToString() { return $this.Render() }

}

class Element

{

[string] $Tag

[string] $Text

[hashtable] $Attributes

[string] Render() {

$attributesText= ""

if ($Attributes)

{

foreach ($attr in $Attributes.Keys)

{

#BUGBUG - need to validate keys against attribute

$attributesText = " $attr=`"$($Attributes[$attr])`""

}

}

return "<${tag}${attributesText}>$text</$tag>`n"

}

[string] ToString() { return $this.Render() }

}

#

# Helper functions for creating specific element types on top of the classes.

# These are required because types aren’t visible outside of the module.

#

function H1 { [Element] @{ Tag = "H1" ; Text = $args.foreach{$\_} -join " " }}

function H2 { [Element] @{ Tag = "H2" ; Text = $args.foreach{$\_} -join " " }}

function H3 { [Element] @{ Tag = "H3" ; Text = $args.foreach{$\_} -join " " }}

function P { [Element] @{ Tag = "P" ; Text = $args.foreach{$\_} -join " " }}

function B { [Element] @{ Tag = "B" ; Text = $args.foreach{$\_} -join " " }}

function I { [Element] @{ Tag = "I" ; Text = $args.foreach{$\_} -join " " }}

function HREF

{

param (

$Name,

$Link

)

return [Element] @{

Tag = "A"

Attributes = @{ HREF = $link }

Text = $name

}

}

function Table

{

param (

[Parameter(Mandatory)]

[object[]]

$Data,

[Parameter()]

[string[]]

$Properties = "\*",

[Parameter()]

[hashtable]

$Attributes = @{ border=2; cellpadding=2; cellspacing=2 }

)

$bodyText = ""

# Add the header tags

$bodyText += $Properties.foreach{TH $\_}

# Add the rows

$bodyText += foreach ($row in $Data)

{

TR (-join $Properties.Foreach{ TD ($row.$\_) } )

}

$table = [Element] @{

Tag = "Table"

Attributes = $Attributes

Text = $bodyText

}

$table

}

function TH { ([Element] @{ Tag = "TH" ; Text = $args.foreach{$\_} -join " " }) }

function TR { ([Element] @{ Tag = "TR" ; Text = $args.foreach{$\_} -join " " }) }

function TD { ([Element] @{ Tag = "TD" ; Text = $args.foreach{$\_} -join " " }) }

function Style

{

return [Element] @{

Tag = "style"

Text = "$args"

}

}

# Takes a hash table, casts it to and HTML document

# and then returns the resulting type.

#

function Html ([HTML] $doc) { return $doc }

### Known issues

In this release, there are the following known issues with class-defined DSC resource providers.

* Get-DSCResource does not discover class-defined resources yet.
* The DSC engine caches resources that are implemented as Windows PowerShell modules for efficiency purposes. When you are authoring and testing a resource simultaneously, set the value of DebugMode on the DSC Local Configuration Manager to **True** to force the engine to reload the DSC resource.

There are the following known issues with creating new types.

* New-Object does not work with Windows PowerShell classes in this release. As a workaround, use the static new method; for example, [Point]::new().
* **ReadOnly** is not yet supported.

## Register a PSRepository with PowerShellGet

In this release, it is much easier to configure PowerShellGet to operate against internal repositories. This is done by the following additions:

* Register-PSRepository: Registers a repository for the current user.
* Unregister-PSRepository: Removes a registered repository for the current user.
* Set-PSRepository: Set values for a registered repository.
* Get-PSRepository: Get all registered repositories for the current user.

After a repository is registered, you can use Find-Module and Install-Module to work with it.

#Register a default repository

Register-PSRepository –Name DemoRepo –SourceLocation “https://www.myget.org/F/powershellgetdemo/api/v2” –PublishLocation “<https://www.myget.org/F/powershellgetdemo/api/v2>/package” –InstallationPolicy –Trusted

#Get all of the registered repositories

Get-PSRepository

Name SourceLocation

---- --------------

PSGallery https://msconfiggallery.cloudapp...

DemoRepo https://www.myget.org/F/powershe...

#Search only the new repository for modules

Find-Module -Repository DemoRepo

Repository Version Name

---------- ------- ----

DemoRepo 1.0.1 xActiveDirectory

DemoRepo 1.1.1 SomeModule

#By default, PowerShellGet operates against all registered repositories when none is specified. In this example, the “SomeModule” module is installed from the DemoRepo.

Install-Module SomeModule

#Removing a repository

Unregister-PSRepository DemoRepo

## Network Switch management with Windows PowerShell (improvements)

The Get-NetworkSwitchEthernetPort cmdlet now returns the following additional information with instances:

* IPAddress – the IP address associated with the port
* PortMode – the port mode: access, route or trunk
* AccessVLAN – the ID of the VLAN associated with this port in access mode
* TrunkedVLANList – a list of IDs of VLANs associated with this port in trunk mode

### Fundamental network switch management with Windows PowerShell

The Network Switch cmdlets, introduced in the first WMF 5.0 Preview, enable you to apply switch, virtual LAN (VLAN), and basic Layer 2 network switch port configuration to Windows Server 2012 R2 logo-certified network switches. Microsoft remains committed to supporting the [Datacenter Abstraction](http://technet.microsoft.com/en-us/cloud/dal.aspx) Layer (DAL) vision, and to show value for our customers and partners in this space. Using these cmdlets you can perform:

* + - Global switch configuration, such as:
      * Sett host name
      * Set switch banner
      * Persist configuration
      * Enable or disable feature
    - VLAN configuration:
      * Create or remove VLAN
      * Enable or disable VLAN
      * Enumerate VLAN
      * Set friendly name to a VLAN
    - Layer 2 port configuration:
      * Enumerate ports
      * Enable or disable ports
      * Set port modes and properties
      * Add or associate VLAN to Trunk or Access on the port

Start exploring by looking for all of the NetworkSwitch cmdlets!

PS> Get-Command \*-NetworkSwitch\*

|  |  |  |
| --- | --- | --- |
| CommandType | Name | Source |
| ----------- | ---- | ------ |
| Function | Disable-NetworkSwitchEthernetPort | NetworkSwitch |
| Function | Disable-NetworkSwitchFeature | NetworkSwitch |
| Function | Disable-NetworkSwitchVlan | NetworkSwitch |
| Function | Enable-NetworkSwitchEthernetPort | NetworkSwitch |
| Function | Enable-NetworkSwitchFeature | NetworkSwitch |
| Function | Enable-NetworkSwitchVlan | NetworkSwitch |
| Function | Get-NetworkSwitchEthernetPort | NetworkSwitch |
| Function | Get-NetworkSwitchFeature | NetworkSwitch |
| Function | Get-NetworkSwitchGlobalData | NetworkSwitch |
| Function | Get-NetworkSwitchVlan | NetworkSwitch |
| Function | New-NetworkSwitchVlan | NetworkSwitch |
| Function | Remove-NetworkSwitchEthernetPortIPAddress | NetworkSwitch |
| Function | Remove-NetworkSwitchVlan | NetworkSwitch |
| Function | Restore-NetworkSwitchConfiguration | NetworkSwitch |
| Function | Save-NetworkSwitchConfiguration | NetworkSwitch |
| Function | Set-NetworkSwitchEthernetPortIPAddress | NetworkSwitch |
| Function | Set-NetworkSwitchPortMode | NetworkSwitch |
| Function | Set-NetworkSwitchPortProperty | NetworkSwitch |
| Function | Set-NetworkSwitchVlanProperty | NetworkSwitch |

More information is available in Jeffrey Snover’s WMF 5.0 Preview announcement blog: <http://blogs.technet.com/b/windowsserver/archive/2014/04/03/windows-management-framework-v5-preview.aspx>

# Scenarios enabled by a previous release of WMF 5.0 Preview

## Discover and install Windows PowerShell modules with PowerShellGet

PowerShellGet is a new way to discover, install, and update Windows PowerShell Modules. New in WMF 5.0 Preview May 2014, PowerShellGet contains a set of cmdlets that enable users to interact with an online module gallery.

Using PowerShellGet, you can:

* Discover modules from the gallery by running the Find-Module command
* Install modules from the gallery by running the Install-Module command
* Update installed modules by running the Update-Module command

Get started by running Find-Module to see all available modules!

PS> Find-Module

## Discover and install software with OneGet

OneGet is a new way to discover and install software packages from the web, and was introduced in the first WMF 5.0 Preview.

With OneGet, you can:

* Manage a list of software repositories in which packages can be searched, acquired, and installed
* Search and filter your repositories to find the packages you need
* Seamlessly install and uninstall packages from one or more repositories with a single Windows PowerShell command

This first version of OneGet installs and searches software from Chocolatey repositories. Support of additional repositories will come in subsequent versions.

Start exploring by importing the OneGet module!

PS> Import-Module -Name OneGet

More information is available in Jeffrey Snover’s WMF 5.0 Preview announcement blog: <http://blogs.technet.com/b/windowsserver/archive/2014/04/03/windows-management-framework-v5-preview.aspx>

# Functionality that is still in early stages of development

## Centralized DSC Error reporting

Rich error information is not only logged in the event log, but it can be sent to a central location for later analysis. Administrators would be able to use this central location to view all errors that have occurred for any server in their environment. After the report server is defined in the meta-configuration, all errors are sent to the report server, and then stored in a database. You can set up this functionality regardless of whether or not the target node is configured to pull its configuration from a pull server.

Some artifacts from this scenario are visible, but are not expected to be in a supported state.

# Known issues

## Reverting to a Windows 8.1 (2012 R2) build from WMF 5.0 Preview can break DSC Cmdlets because of updates in metaconfig.mof

Resolution: Delete metaconfig.mof.

1. Open **powershell.exe** with elevated user rights (run as administrator).
2. Run the following command in the console:
   1. Remove-Item -Path $env:SystemRoot\system32\Configuration\metaconfig.mof

## WMF 5.0 Preview installation appears to succeed (or fails), but the installation rolls back after the system is restarted

Resolution: Delete the **\\root\microsoft\windows\desiredstateconfiguration** namespace in WMI.

1. Open **powershell.exe** with elevated user rights (run as administrator).
2. Run the following commands:
   1. $dscNamespace = Get-CimInstance -Namespace root\microsoft\windows -Query "select \* from \_\_namespace where name = 'desiredstateconfiguration'"
   2. $dscNamespace | Remove-CimInstance
   3. mofcomp.exe %windir%\system32\wbem\DSCCoreConfProv.mof
3. Install the WMF 5.0 Preview package.

## WMF 5.0 Preview uninstallation fails during restart step

When you are uninstalling WMF 5.0 Experimental Release July 2014, uninstallation can fail on some computers, and revert during the restart step of uninstallation.

Resolution: Delete the [**\\root\microsoft\windows\desiredstateconfiguration**](file://root/microsoft/windows/desiredstateconfiguration) namespace in WMI before uninstalling WMF 5.0 Experimental Release July 2014.

1. Open **powershell.exe** with elevated user rights (**run as administrator**).
2. Run the following commands:

$dscNamespace = Get-CimInstance -Namespace root\microsoft\windows -Query "select \* from \_\_namespace where name = 'desiredstateconfiguration'"

$dscNamespace | Remove-CimInstance

# Known incompatibilities

**Systems that are running the following server applications should not run Windows Management Framework 5.0 at this time.**

* + - System Center 2012 Configuration Manager (not including SP1)
    - Windows Small Business Server 2011 Standard

**Servers that are running the following applications can now run WMF 5.0. The versions listed are the minimum releases required to run WMF 5.0.**

* + - Microsoft Exchange Server 2013 [Service Pack 1](http://go.microsoft.com/fwlink/?LinkId=389952)
    - Microsoft SharePoint Server 2013 [Service Pack 1](http://go.microsoft.com/fwlink/?LinkId=389955)

After the updates are applied, WMF 5.0 is supported on servers that are running these applications. WMF 5.0 is otherwise supported on Windows Server 2012 R2, Windows 8.1 Pro, and Windows 8.1 Enterprise.

# Feedback and contact

For issues or feedback you would like to report to us, use our Connect website:

<https://connect.microsoft.com/PowerShell/Feedback>

# Additional links

WMF 5.0 Preview May 2014: <http://www.microsoft.com/en-us/download/details.aspx?id=42316>