wINDOWS MANAGEMEnt Framework 4.0

Release Notes

# Overview

Windows Management Framework (WMF) 4.0 contains functionality that has been updated from WMF 3.0, and is available for installation on Windows 7 with Service Pack 1 (SP1), Windows Server 2008 R2 with SP1, and Windows Server 2012. WMF 4.0 contains updated versions of the following features:

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

Windows PowerShell 4.0 includes a new feature, also available in WMF 4.0:

* Windows PowerShell Desired State Configuration (DSC)

To use this updated management infrastructure to manage Windows 7 SP1, Windows Server 2008 R2 SP1, and Windows Server 2012, Windows Management Framework 4.0 must be installed on computers that are running the older operating systems. Windows Management Framework 4.0 cannot be installed on Windows 8. However, you can obtain updated functionality included in WMF 4.0 by installing [Windows 8.1](http://windows.microsoft.com/windows-8/preview), which is available as a free update for Windows 8.

# Requirements

For this Release, WMF 4.0 installs only on the following operating systems:

|  |  |  |
| --- | --- | --- |
| Operating System | Service Pack Level | Editions |
| Windows 7 | [Service Pack 1](http://windows.microsoft.com/installwindows7sp1) | All |
| Windows Server 2008 R2 | [Service Pack 1](http://www.microsoft.com/en-us/download/details.aspx?id=5842) | All except IA64 |
| Windows Server 2012 |  | All except IA64 |
| Windows Embedded 7 |  | All |

WMF 3.0 and the Windows PowerShell 2.0 engine are not required to install WMF 4.0. However, the Windows PowerShell 2.0 engine can be installed separately to run scenarios that specifically use Windows PowerShell 2.0 functionality, such as Powershell.exe -Version, Start-Job -PSVersion, and Register-PSSessionConfiguration -PSVersion. The Windows PowerShell 2.0 engine is not included with WMF 4.0. It can be installed by using Server Manager or Control Panel.

On Windows Server 2012, Windows 8, or by running WMF 3.0, you can choose to run either Windows PowerShell 2.0 or Windows PowerShell 3.0. After installing WMF 4.0, you can run either Windows PowerShell 2.0 or Windows PowerShell 4.0. If you try to run Windows PowerShell 3.0 after installing WMF 4.0, the Windows PowerShell version that runs is 4.0. Windows PowerShell 4.0 is fully backwards-compatible with Windows PowerShell 3.0, except as noted in the Breaking Changes section of this document.

[.NET Framework 4.5](http://www.microsoft.com/download/details.aspx?id=30653) must be installed before installing Windows Management Framework 4.0.

On Windows Server 2008 R2, Windows PowerShell ISE must be installed separately from WMF 4.0. To install Windows PowerShell ISE, run the Add Features Wizard in Server Manager to add the optional Windows PowerShell ISE feature.

Install the latest Windows updates before installing WMF 4.0.

# How to Install and Uninstall WMF 4.0

## Installation Instructions

### Windows Server 2012

Ensure all of the following prerequisites have been met:

* **Windows Server 2012**
* **Microsoft** [**.NET Framework 4.5**](http://www.microsoft.com/download/details.aspx?id=30653)
* **Windows PowerShell 2.0 (if applicable)**
* Windows PowerShell 2.0 is disabled by default on Windows Server 2012 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**.

### Windows 7 or Windows Server 2008 R2 (with Service Pack 1)

Ensure all of the following prerequisites have been met:

* **Windows 7 with Service Pack 1** or **Windows Server 2008 R2 with Service Pack 1**
* **Microsoft** [**.NET Framework 4.5**](http://www.microsoft.com/download/details.aspx?id=30653)
* **Windows PowerShell 2.0 (if applicable)**
* Windows PowerShell 2.0 is disabled by default on Windows Server 2008 R2 SP1 Server Core. It can be enabled by running the following two commands:

dism /online /enable-feature:MicrosoftWindowsPowerShell

dism /online /enable-feature:MicrosoftWindowsPowerShell-WOW64

Installation:

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

**Note:** When you run the WMF 4.0 installation package, the following updates are installed first:

* [KB 2823180](http://support.microsoft.com/kb/2823180), which is the equivalent of the [Windows Management Framework 3.0 Update](http://support.microsoft.com/kb/2823180). This update is not removed if you uninstall WMF 4.0.
* [KB 2872047](http://go.microsoft.com/fwlink/?LinkId=328210) if WMF 3.0 is already installed on the computer.
* [KB 2872035](http://go.microsoft.com/fwlink/?LinkID=328211) if WMF 3.0 is not already installed on the computer.

#### Enabling Workflow validation

Workflow validation is disabled by default for the Microsoft.PowerShell.Workflow session configuration on Windows Server 2008 R2 SP1. This change only impacts workflows that are running in the default workflow session configuration.

The result of this change is that if the session configuration defines a list of disallowed activities, and one of those activities is requested, the activity is invoked, and no warnings are reported. This also prevents the workflow from being suspended.

To work around this issue, run the following commands, at least once in any session that is used to start or resume a workflow:

$config = Get-PSSessionConfiguration Microsoft.PowerShell.Workflow

$config.EnableValidation = $true

Set-PSSessionConfiguration $config

#### Enabling Windows PowerShell Desired State Configuration

Windows PowerShell Desired State Configuration (DSC) uses Windows Remote Management (WinRM) to communicate with remote devices. To use DSC, you must be sure that Windows Remote Management is enabled. Windows Remote Management is enabled by default on Windows Server 2012, but it is not enabled by default on Windows 8, Windows 7, or Windows Server 2008 R2. To enable Windows Remote Management, after you have installed WMF 4.0, do the following:

1. Run Windows PowerShell as Administrator.
2. Run **[Set-WSManQuickConfig](http://technet.microsoft.com/en-us/library/hh849867(v=wps.640).aspx)**.

For more information about Windows Remote Management, see [Windows Remote Management](http://msdn.microsoft.com/library/aa384426(v=vs.85).aspx) on MSDN.

**Note:** Windows PowerShell Desired State Configuration is not supported on 32-bit operating systems.

### CONFIGURING THE Windows PowerShell Desired State Configuration Service

To set up a server as a Windows PowerShell Desired State Configuration pull server, additional steps are needed:

On Windows Server 2012:

* Install Windows PowerShell Desired State Configuration Service (DSC-Service) by running the **Install-WindowsFeature** cmdlet,or the Add Roles and Features Wizard in Server Manager.
* This installs all dependent features, such as Web Server (IIS) (Web-Server), Management OData IIS Extension (ManagementOdata), etc.

On Windows Server 2008 R2 SP1:

1. Install Web Server (IIS), HTTP Tracing, Management OData IIS Extension (ManagementOdata), WCF Http Activation, and ASP.NET.

Add-WindowsFeature Web-Server

Add-WindowsFeature Web-Http-Tracing

dism /online /enable-feature:ManagementOdata

Add-WindowsFeature NET-Http-Activation

Add-WindowsFeature Web-ASP-NET

1. Install Windows PowerShell Desired State Configuration Service (DSC-Service) feature by running **dism.exe**.

dism /online /enable-feature:DSC-Service

On either Windows Server 2012 or Windows Server 2008 R2 SP1, create a Windows PowerShell Web Services endpoint by running the setup script (SetupIISConfig.ps1) from the Management OData IIS Extension samples at the following location: http://code.msdn.microsoft.com/windowsdesktop/PswsRoleBasedPlugins-1c7a7ef1/sourcecode?fileId=42767&pathId=1331822308.

Additional files required to set up the endpoint can be found in $pshome\Modules\PSDesiredStateConfiguration\PullServer, after installing Windows PowerShell Desired State Configuration Service.

**Note:** To finish configuring the endpoint and enable its functionality, you might be required to run the .NET Framework 4.5 installer again.

### Known Issue: Partial installation without .NET Framework 4.5

When you attempt to install WMF 4.0 on Windows 7 or Windows Server 2008 R2 without .NET Framework 4.5, only the prerequisite QFEs that are included in the package are installed. The installation process attempts to install WMF 4.0, but fails. The two prerequisite QFEs remain on the computer after WMF 4.0 installation fails.

To repair your WMF 4.0 installation after this failure, install .NET Framework 4.5, and then run the WMF 4.0 MSU installation package again to install WMF 4.0. The installation process skips the QFEs, and installs WMF 4.0.

## 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 4.0** from the list of installed updates. On Windows 7, the Windows Management Framework 4.0 update corresponds to KB2819745, while on Windows Server 2012, this corresponds to KB2799888. Click **Uninstall.**

### By using Command Prompt

1. Open **Command Prompt**.
2. Run the following command for Windows 7 or Windows Server 2008 R2:

wusa /uninstall /kb:2819745

Run the following command for Windows Server 2012:

wusa /uninstall /kb:2799888

### WMI Event Logs Removed On Uninstallation

When Windows Management Framework 4.0 is uninstalled, some WMI event logs are erroneously removed from Event Viewer. The affected event logs include the following:

* WMI-Activity Debug
* WMI-Activity Operational
* WMI-Activity Trace

To work around this issue, back up the following registry keys before installing WMF 4.0, and then replace the registry keys after you uninstall WMF 4.0:

* HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels\Microsoft-Windows-WMI-Activity\Debug
* HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels\Microsoft-Windows-WMI-Activity\Operational
* HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Channels\Microsoft-Windows-WMI-Activity\Trace

## Upgrade Scenarios

After installing Windows Management Framework 4.0, it is possible to upgrade your operating system to a newer release of Windows, such as from Windows 7 to Windows 8.

After upgrading, WMF 4.0 is no longer expected to be installed on a computer.

However, during testing of these scenarios, we discovered several issues, described in this section. There are some issues where an additional file remains after the operating system upgrade, despite the absence of WMF 4.0. Additionally, there are issues listed below with functional impact. These issues were not found when uninstalling WMF 4.0 before performing an operating system upgrade.

We highly recommend uninstalling WMF 4.0 before upgrading the computer’s operating system.

### Upgrading From Windows 7 to Windows 8 after installing WMF 4.0 Removes registry keys for Windows PowerShell 1.0 Providers

Upgrading from Windows 7 to Windows 8 after installing WMF 4.0 clears the custom entries within HKEY\_CLASSES\_ROOT.

Entries that are added by registering a Windows PowerShell 1.0 provider are no longer shown under the key, and thus do not work properly.

Note that the providers themselves are not lost as part of this issue.

To work around this issue:

* **Before upgrading to the newer operating system**, uninstall Windows Management Framework 4.0.

# Changes in Windows Management Framework 4.0

### Windows PowerShell

Windows PowerShell is a task-based command-line shell and scripting language designed especially for system administration. Built on the .NET Framework, Windows PowerShell helps IT professionals and power users control and automate the administration of the Windows operating system and applications that run on Windows.

Windows PowerShell allows you to run scripts, functions, and modules of cmdlets. Cmdlets are simple verb-noun commands that help you automate management of roles and features that run on the Windows operating system.

After you install WMF 4.0, Windows PowerShell is upgraded to version 4.0.

#### Versioning Notes

The following versioning-related items have changed in WMF 4.0.

* Version 4.0 of Windows PowerShell replaces version 3.0. Requests to load version 3.0 automatically forward to load version 4.0. For example, running Powershell.exe -Version 3.0 automatically loads version 4.0.
* Session configurations that load Windows PowerShell 3.0 automatically load version 4.0.
* Cmdlets that load a specific version of Windows PowerShell automatically use version 4.0 when version 3.0 is specified. This includes cmdlets such as Start-Job, Set-PSSessionConfiguration, and Register-PSSessionConfiguration.
* The PowerShellVersion registry value at HKLM:\SOFTWARE\Microsoft\PowerShell\3 has changed from 3.0 to 4.0.
* The PSCompatibleVersion registry value at HKLM:\SOFTWARE\Microsoft\PowerShell\3 has changed from 1.0, 2.0, 3.0 to 1.0, 2.0, 3.0, 4.0.
* The value of $PSVersionTable.PSVersion is changed from 3.0 to 4.0.

The following versioning-related items have not changed in WMF 4.0.

* The location of $pshome is %windir%\system32\WindowsPowerShell\v1.0
* Windows PowerShell script files have a .ps1 file extension
* Windows PowerShell registry keys are located at HKLM:\SOFTWARE\Microsoft\PowerShell\1 and HKLM:\SOFTWARE\Microsoft\PowerShell\3

#### Important Changes in Windows PowerShell for WMF 4.0

**Change:** Behavior of DefaultCommandPrefix

**Description:** In Windows PowerShell 3.0, if a module uses the DefaultCommandPrefix key in its manifest or if the user imports a module with the Prefix parameter, the ExportedCommands property of the module shows the commands in the module without the prefix. The commands can also be run without the prefix using module-qualified syntax: ModuleName\CommandName

In Windows PowerShell 4.0, if a module uses the DefaultCommandPrefix key in its manifest, or if the user imports a module with the Prefix parameter, the ExportedCommands property of the module shows the commands in the module with the prefix. When you run the commands by using the module-qualified syntax ModuleName\CommandName, the command names must include the prefix.

**Breaking?** Yes

**Example:** Suppose SampleModule defines the DefaultCommandPrefix in its manifest with the prefix "Abc" and exports the Get-SampleCommand cmdlet.

In Windows PowerShell 3.0:

C:\> $m = Get-Module SampleModule -ListAvaiable

C:\> $m.ExportedCommands | Format-List

Key : Get-SampleCommand

Value : Get-SampleCommand

In Windows PowerShell 4.0:

C:\> $m = Get-Module SampleModule -ListAvailable

C:\> $m.ExportedCommands | Format-List

Key : Get-AbcSampleCommand

Value : Get-AbcSampleCommand

In Windows PowerShell 3.0:

C:\> Import-Module SampleModule

C:\> SampleModule\Get-SampleCommand

<no error - command is executed>

C:\> SampleModule\Get-AbcSampleCommand

<CommandNotFoundException>

In Windows PowerShell 4.0:

C:\> Import-Module SampleModule

C:\> SampleModule\Get-SampleCommand

<CommandNotFoundException>

C:\> SampleModule\Get-AbcSampleCommand

<no error - command is executed>

**Workaround:** When you are invoking commands with prefixes by using module-qualified syntax, include the prefix in the command name.

|  |  |
| --- | --- |
|  |  |

**Change:** Windows PowerShell version change from 3.0 to 4.0

**Description:** The Windows PowerShell version is changing from 3.0 to 4.0. This change is visible in the following locations:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\PowerShell\3\PowerShellEngine\PowerShellVersion

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\PowerShell\3\PowerShellEngine\PSCompatibleVersions

$PSVersionTable.PSVersion

**Breaking?** Yes

Code or scripts that include a hard-coded check for Windows PowerShell version 3.0 (either by using the registry keys identified in the preceding paragraphs, or by using **$PSVersionTable.PSVersion**) are affected by the Windows PowerShell version change from 3.0 to 4.0.

Unless you specifically hard-code an exact Windows PowerShell version check in code or script, you should not be affected. Scripts and cmdlets that are written for Windows PowerShell 3.0 continue to run with no changes required in Windows PowerShell 4.0, by default.

**Action Required:** If you are affected, consider changing your version check to a minimum version check instead of an exact version check. If you must have an exact version check, change the logic in your check from 3.0 to 4.0, to run your code in Windows PowerShell 4.0.

|  |  |
| --- | --- |
|  |  |

**Change:** Behavior of Save-Help

**Description:** In Windows PowerShell 3.0, **Save-Help** worked only for modules that are installed on the local computer. Although it was possible to import a module from a remote computer, or obtain a reference to a **PSModuleInfo** object from a remote computer by using Windows PowerShell remoting, the **HelpInfoUri** property was not preserved, and **Save-Help** would not work for the remote module’s Help. In Windows PowerShell 4.0, the **HelpInfoUri** property is preserved over Windows PowerShell remoting, which enables **Save-Help** to work for remote modules. It is also possible to save a **PSModuleInfo** object to disk or removable media using **Export-CliXml** on a computer without Internet access, import the object on a computer with Internet access, and run **Save‑Help** on the deserialized **PSModuleInfo** object. The saved help can be transported using removable media back to the original computer and installed by running **Update-Help**. This process can be used to install help on computers without any kind of network access.

**Breaking?** No

**Example 1:** Run **Save-Help** to save the help for the **DhcpServer** module from an Internet-connected client computer, without installing the **DhcpServer** module or DHCP Server role on the local computer.

# Option 1: Run Invoke-Command to get the remote module and call Save-Help

$m = Invoke-Command -ComputerName RemoteServer -ScriptBlock { Get‑Module ‑Name DhcpServer -ListAvailable }

Save-Help -Module $m -DestinationPath C:\SavedHelp

# Option 2: Use a PSSession to get the remote module and call Save-Help

$s = New-PSSession -ComputerName RemoteServer

$m = Get-Module -PSSession $s -Name DhcpServer -ListAvailable

Save-Help -Module $m -DestinationPath C:\SavedHelp

# Option 3: Use a CimSession to get the remote module and call Save-Help

$c = New-CimSession -ComputerName RemoteServer

$m = Get-Module -CimSession $c -Name DhcpServer -ListAvailable

Save-Help -Module $m -DestinationPath C:\SavedHelp

**Example 2:** Install help for the **DhcpServer** module on a computer without any network access.

# Run Export-CliXml to serialize the PSModuleInfo object to disk or removable media

$m = Get-Module -Name DhcpServer –ListAvailable

Export-CliXml –Path E:\UsbFlashDrive\DhcpModule.xml –InputObject $m

# Transport the removable media to a computer with Internet access and import with Import-CliXml

$deserialized\_m = Import-CliXml E:\UsbFlashDrive\DhcpModule.xml

Save-Help -Module $deserialized\_m -DestinationPath E:\UsbFlashDrive\SavedHelp

# Transport the removable media back to the computer without network access and install the help

Update-Help –Module DhcpServer –SourcePath E:\UsbFlashDrive\SavedHelp

|  |  |
| --- | --- |
|  |  |

**Change:** Windows PowerShell debugger enhancements to support Windows PowerShell Workflow debugging and improved remote session experience

**Description:** The Windows PowerShell debugger has been enhanced to allow debugging of Windows PowerShell workflows, as well as scripts that are running on remote computers. Windows PowerShell workflows can now be debugged at the script level from either the Windows PowerShell command line or Windows PowerShell ISE. Windows PowerShell scripts, including script workflows, can now be debugged over remote sessions. Remote debugging sessions are preserved over Windows PowerShell remote sessions that are disconnected and then later reconnected.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** RunNow parameter added to Register-ScheduledJob and Set-ScheduledJob

**Description:** A RunNow parameter for Register-ScheduledJob and Set-ScheduledJob eliminates the need to set an immediate start date and time for jobs by using the Trigger parameter.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Support to set Headers in Invoke-RestMethod and Invoke-WebRequest

**Description:** Invoke-RestMethod and Invoke-WebRequest now let you set all headers by using the Headers parameter. Although this parameter has always existed, it was one of several parameters for the web cmdlets that resulted in exceptions or errors.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** FullyQualifiedName Parameter added to Get-Module

**Description:** Get-Module has a new parameter, FullyQualifiedName, of the type ModuleSpecification[]. The Name parameter of Get-Module now lets you specify a module by using the module’s name, version, and GUID. As before, it also lets you specify a module by using only the name.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Default execution policy setting on Windows Server 2012 R2 is RemoteSigned

**Description:** The default execution policy setting on Windows Server 2012 R2 is **RemoteSigned**. On Windows 8.1, there is no change in the default setting, which is **Restricted**.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Support for method invocation by dynamic method names

**Description:** Starting in Windows PowerShell 4.0, method invocation by using dynamic method names is supported.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Asynchronous workflow jobs are no longer deleted when the time-out period that is specified by the PSElapsedTimeoutSec workflow common parameter has elapsed.

**Description:** Asynchronous workflow jobs are no longer deleted when the time-out period that is specified by the PSElapsedTimeoutSec workflow common parameter has elapsed.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** RepeatIndefinitely parameter added to New-JobTrigger and Set-JobTrigger

**Description:** A new parameter, RepeatIndefinitely, has been added to the New-JobTrigger and Set-JobTrigger cmdlets. This eliminates the necessity of specifying a TimeSpan.MaxValue value for the RepetitionDuration parameter to run a scheduled job repeatedly, for an indefinite period.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** PassThru parameter added to Enable-JobTrigger and Disable-JobTrigger

**Description:** A PassThru parameter has been added to the Enable-JobTrigger and Disable-JobTrigger cmdlets. The PassThru parameter displays any objects that are created or modified by your command.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** WorkgroupName used in both Add-Computer and Remove-Computer

**Description:** The parameter names for specifying a workgroup in the Add-Computer and Remove-Computer cmdlets are now consistent. Both cmdlets now use the parameter WorkgroupName.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** PipelineVariable added as a common parameter

**Description:** A new common parameter, PipelineVariable, has been added. PipelineVariable lets you save the results of a piped command (or part of a piped command) as a variable that can be passed through the remainder of the pipeline.

Support of this parameter extends to the context of iterative pipelines, such as those used by System Center Orchestrator; that is, pipelines that run commands simply left-to-right, as opposed to interspersed running by using streaming.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Collection filtering support

**Description:** Collection filtering by using a method syntax is now supported.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** IncludeUserName added to Get-Process

**Description:** The Get-Process cmdlet has a new switch parameter, IncludeUserName.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** New cmdlet Get-FileHash

**Description:** A new cmdlet, Get-FileHash, that gets information about file hashes, has been added.

**Breaking?** No

### Windows PowerShell Workflow

**Change:** Parameter binding enhanced to work outside of tab completion scenarios

**Description:** Parameter binding has been significantly enhanced to work outside of tab completion scenarios, such as with commands that do not exist in the current runspace.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Support for custom container activities

**Description:** Support for custom container activities has been added to Windows PowerShell Workflow. If an activity parameter is of the types Activity, Activity[], or is a generic collection of activities, and the user has supplied a script block as an argument, then Windows PowerShell Workflow converts the script block to XAML, as with normal Windows PowerShell script-to-workflow compilation.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Automatic reconnection to managed nodes after a crash

**Description:** After a crash, Windows PowerShell Workflow automatically reconnects to managed nodes.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** ThrottleLimit support in ForEach -Parallel

**Description:** You can now throttle ForEach -Parallel activity statements by using the ThrottleLimit property.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Suspend parameter added to ErrorAction

**Description:** The ErrorAction common parameter has a new valid value, **Suspend**, that is exclusively for workflows.

**Breaking?** No

|  |  |
| --- | --- |
|  |  |

**Change:** Workflow endpoints automatically close when they are no longer in use

**Description:** A workflow endpoint now automatically closes if there are no active sessions, no in-progress jobs, and no pending jobs. This feature conserves resources on the computer that is acting as the workflow server, when the automatic closure conditions have been met.

**Breaking?** No

### Windows PowerShell ISE

Windows PowerShell Integrated Scripting Environment (ISE) is a graphical host application for Windows PowerShell. Windows PowerShell ISE lets you run commands, and write, edit, run, test, and debug scripts in an environment that displays syntax in colors and that supports Unicode.

Windows PowerShell ISE is designed for users at all levels of proficiency. Beginners will appreciate the syntax colors and the context-sensitive Help. Multiline editing makes it easy to try the examples that you copy from the Help topics and from other sources. Advanced users will appreciate the availability of multiple execution environments, the built-in debugger, and the extensibility of the Windows PowerShell ISE object model.

Major feature changes in Windows PowerShell ISE in WMF 4.0 include the following:

* Support for Windows PowerShell Workflow debugging
* Support for remote script debugging
* IntelliSense support for Windows PowerShell Desired State Configuration providers and configurations

Changes to Windows PowerShell ISE in WMF 4.0 are minor bug fixes, including fixes for the following issues:

* Certain hotkeys were not localized
* IntelliSense window had black text on black foreground in high contrast mode
* Show-Command was incorrectly prompting the user for input
* F1 help was not properly escaping input
* AutoSave information that is not valid was created, which prevented ISE from starting
* Ctrl+A did not open the File dialog box in Spanish-localized versions of Windows
* ISE did not close properly when Invoke-WebRequest was called
* Some help links were not working correctly

### Windows PowerShell Desired State Configuration (DSC)

#### Overview

Windows PowerShell Desired State Configuration (DSC) helps ensure that the resources in your datacenter are correctly configured. DSC is a set of Windows PowerShell language extensions and providers that enable declarative, autonomous, and idempotent (repeatable) deployment, configuration, and conformity of datacenter resources. DSC enables an IT Pro, developer, or fabric administrator to define the configuration of target nodes (computers or devices) and prevent configuration inconsistencies or "drift".

Customers must ensure that their computers and other managed devices are in a desired state. Although there are many different tools and scripting languages to accomplish configuration consistency as required in different environments, no existing tools allow this across all devices. Windows PowerShell Desired State Configuration allows all devices to be managed by using Windows PowerShell.

For more information about using DSC, see [Windows PowerShell Desired State Configuration Overview](http://technet.microsoft.com/library/dn249912.aspx) on the Microsoft TechNet website.

##### Local Configuration Manager

The Local Configuration Manager is the part of the DSC system that receives, coordinates, and implements configuration data on the target node.

##### Windows Powershell language extensions

DSC extends the Windows PowerShell language to support importing industry-standard Managed Object Format (.mof) schema files that are converted to Windows PowerShell keywords. The keywords are used to describe the desired configuration of resources in the data center. The new DSC keywords include the following:

**Keyword:** Configuration

**Description:** Used to define the desired configuration

**Syntax:** 'configuration' <SingleNameExpression> '{' <statementList> '}'

**Keyword:** Node

**Description:** Used to define the desired configuration for one or more nodes.

**Syntax:** 'node' <SingleNameExpression> '{' <statementList> '}'

##### DSC cmdlets

In addition to new language keywords, DSC includes the following cmdlets for managing configurations:

**Start-DscConfiguration:** Deploys a configuration to one or more target nodes, and applies the configuration on those nodes by using the local configuration manager.

**Get-DscConfiguration:** Returns the current configuration from one or more target nodes.

**Test-DscConfiguration:** Checks one or more target nodes, and returns a Boolean value indicating whether the current desired state matches the actual state.

##### Configuration providers

A collection of configuration providers is part of the core DSC system. These providers help you manage common configuration types, such as files, processes, services, and server roles.

##### Windows PowerShell Desired State Configuration Service

Windows PowerShell Desired State Configuration (DSC) makes it possible to set up a server as a central configuration pull server. You can store configurations of the computers (nodes) in your environment on this pull server. The pull server can also store custom DSC resources that the target nodes need for their configuration. This functionality is useful for environments where there are a large number of target nodes to configure, and where you want your target nodes to both get the right configuration information as they come online, and check periodically for configuration updates.

For more information about how to set up Windows PowerShell Desired State Configuration Serviceon a server, see [Enabling Windows PowerShell Desired State Configuration Service](#_Enabling_Windows_PowerShell).

Note: This feature is not available in the Server Core installation option of Windows Server.

##### File share pull provider

The file share pull provider is an auxiliary module that is used by Local Configuration Manager (LCM) to retrieve configuration MOF files and providers from a file share or a local folder that is different from the default location.

#### Configuration example

To use DSC, first define a desired configuration. Like functions, configurations in DSC can be defined in the Windows PowerShell language by using the Configuration keyword, and stored in script (.ps1) or module (.psm1) files. Also similar to functions, configurations need to be defined and then run. Each Configuration block must have at least one Node block. Each Node block can have one or more resource provider blocks. You can use the same role provider more than once in the same Node block.

***SampleConfiguration.ps1***

configuration MyWebConfig

{

# Parameters are optional

param ($MachineName, $WebsiteFilePath)

# A Configuration block can have one or more Node blocks

node $MachineName

{

# Next, specify one or more resource provider blocks

# WindowsFeature is one of the providers you can use in a Node block

# This example ensures the Web Server (IIS) role is installed

WindowsFeature IIS

{

Ensure = "Present" # To uninstall the role, set Ensure to "Absent"

Name = "Web-Server" # Use the Name property from Get-WindowsFeature

}

# You can use the File provider to create files and folders

# "File" is the name of the resource provider to use

# "WebDirectory" is the name you want to use to refer to this instance

File WebDirectory

{

Ensure = "Present" # You can also set Ensure to "Absent"

SourcePath = $WebsiteFilePath

DestinationPath = "C:\inetpub\wwwroot"

DependsOn = "[WindowsFeature]IIS" # Use DependsOn for dependencies

}

}

}

To use a configuration, invoke the Configuration block the same way you would invoke a Windows PowerShell function, passing in any expected parameters you have defined (two in the sample above). For example, in this case, the MyWebConfig configuration can be invoked as follows:

MyWebConfig -MachineName "TestMachine" –WebsiteFilePath "\\filesrv\WebFiles"

This creates a MOF file known as the **configuration instance document**. You can run it by using the Start-DscConfiguration cmdlet like this:

Start-DscConfiguration -Path .\MyWebConfig –Wait –Verbose

### Windows PowerShell Web Services

Formally called Management OData IIS Extension, Windows PowerShell Web Services (PSWS) exposes OData functionality through a web endpoint by using Windows PowerShell cmdlets and standard HTTP requests.

#### Improved error messages

When an error occurs in PSWS during cmdlet execution, more detailed error messages are returned to the caller. In addition, error codes follow [Windows Azure REST API error code guidelines](http://msdn.microsoft.com/library/windowsazure/dd179357.aspx).

#### Versioning support

An endpoint can now define the API version as well as enforce usage of a specific API version. Whenever version mismatches occur between client and server, errors are stored on both sides.

#### Simplified OData dispatch schema

Management of the dispatch schema has been simplified by automatically generating values for any missing fields in the schema. This generation occurs even if the dispatch schema does not exist.

A Version field within the schema can be used to identify which version a PSWS endpoint is running. If the value of Version equals 1.0, or is the only element missing from the schema, PSWS treats the schema as in previous releases, and does not automatically generate values.

**Note**: The Windows PowerShell Web Services Invoke feature is disabled when the dispatch schema is omitted.

Windows PowerShell Web Services Invoke was a feature enabled in older versions of PSWS to allow users to invoke Windows PowerShell cmdlets remotely by using PSWS.

This feature is enabled if the dispatch schema is partially or fully completed.

#### Support for complex types

Type handling in PSWS has been improved to support types that use a different constructor than the default constructor, by using behavior similar to the [PSTypeConverter](http://msdn.microsoft.com/library/windows/desktop/system.management.automation.pstypeconverter(v=vs.85).aspx) in Windows PowerShell. This enables the usage of complex types with PSWS.

#### Multilevel association support

OData allows expanding an associated instance while running a query. For a typical example of Supplier, Product, and Category, one can query for a supplier and all the products that are supplied by it as follows:

~/Suppliers('Contoso')?$expand=Products

Similarly, a product and its category can be requested:

~/Products('Widgets')?$expand=Category

OData also allows multilevel navigation, as shown in the following example:

~/Suppliers('Contoso')?$expand=Products/Category

Note that filtering occurs before the expansion operation, so you cannot filter based on properties that are found under expanded associations. Paging also occurs before expansion.

#### Large binary stream transfer

Previous to this release, PSWS had existing support for Edm.Binary properties for transferring binary data. This binary data is base64-encoded. The encoding and decoding incurs an extra time cost during transfer. For larger binary contents (such as images, audio, video, etc.), the cost is significant, and it is better to transfer binary data without encoding.

PSWS uses named resource streams for transferring without any encoding. The named resource stream is a property of an entity that is of the Edm.Stream type. Each named resource stream has a separate URI for GET or UPDATE operations.

An HTTP GET request on the entity instance with a named resource stream only returns the URI of the named resource stream. A typical HTTP GET request on a named resource stream is similar to the following:

GET /VirtualHardDrive(guid'6ffddb3c-8f11-4efd-a814-206db4bc4838')/Content

Binary contents can be updated by using HTTP UPDATE/MERGE on the named resource stream. A typical HTTP PUT request is similar to the following:

PUT /VirtualHardDrive(guid'6ffddb3c-8f11-4efd-a814-206db4bc4838')/Content

<binary data>

#### Non-Create, Read, Update, and Delete (CRUD) actions

OData actions provide a mechanism for invoking non-CRUD (or extrinsic) methods on a resource. An action can be invoked by sending an HTTP POST request to the URI that is defined for the action. The parameters for the action are defined in the body of the POST request.

Invoking the action in PSWS results in invoking the cmdlet. This provides an easy way to invoke cmdlets from PSWS that are not CRUD actions.

#### Key As Segment

To be consistent with Azure guidelines, all URLs should be simplified. A change included in Key As Segment allows single keys to be represented as segments. Note that references that use multiple key values require comma-separated values in parenthetical notation, as before.

This change requires the following message in the header of the URL:

context.UrlConventions = DataServiceUrlConventions.KeyAsSegment;

To reduce user effort, a server can have a value in web.config that instructs PSWS to splice the preceding snippet into URL headers before they are parsed.

There are ramifications to supporting Key As Segment, such as ambiguity between keys and actions or properties. Because there is no OData literal format when writing the key value, there is no way to determine if the segment refers to the key value (i.e. is a data token) or to type/action/function name (i.e. is a metadata token). This ambiguity only happens for segments following the collection segments – entity set, collection navigation property, functions that return collection of entities, etc.

To remove the ambiguity, you must specify the '$' segment to indicate that the following segment is a metadata token.

For example, to get all the premium customers from the Customers collection, the URL must look like this:

~/Customers/$/NS.PremiumCustomer

For OData keyword tokens like $count, $metadata, there is no need to add the extra '$' segment, because they start with the '$' character, meaning they are reserved OData tokens.

If the key value starts with the '$' token, it must be escaped with an additional '$' character as follows:

~/Customers/$$count

The preceding URL returns a customer instance with '$count' as the key value, if one exists.

#### Contained resource operations

Before this release of PSWS, the only way to perform Create, Update, or Delete operations was to invoke POST, PUT, or DELETE on a top-level resource. For example:

POST ~/VMRoles

"Contained Resource" operations allows users to achieve the same results while reaching the same resource more indirectly, approaching as if these resources were contained. For example:

POST ~/PhysicalMachines(Key)/Resources(AnotherKey)/VMRoles

Actions can also be performed in a similar manner. Note: POST acts on a collection, while PUT and DELETE act on a single instance.

#### Breaking changes

##### Removal of default sorting for server-driven paging

**Change:** Removal of default sorting for server-driven paging in Management OData IIS Extension

**Description:** In PSWS, default sorting in server-driven paging is no longer supported. Ordering is applied in the PSWS layer only when it is explicitly requested by using **$orderby**.

**What it could break**: A process that requires basic requests to be sorted, and does not perform its own sorting.

**How to fix**: Use **$orderby** explicitly whenever you want to sort.

### WinRM and WMI

Windows Remote Management (WinRM) is the Microsoft implementation of the DMTF standard WS-Management protocol, a Simple Object Access Protocol (SOAP)-based, firewall-friendly protocol that allows hardware and operating systems from different vendors to interoperate.

No new features for WinRM are included in WMF 4.0.

Windows Management Instrumentation (WMI) is the Microsoft implementation of the server and client components of the DMTF standard Common Information Model (CIM) infrastructure. CIM provides a consistent schema and interface for management information and controls.

No new features for WMI are included in WMF 4.0.

### Server Manager WMI Provider

The Server Manager WMI provider allows Server Manager to receive inventory and other data from remote servers.

No new features for the Server Manager WMI provider are included in WMF 4.0.

# Known Incompatibilities

WMF 4.0 has a similar set of known incompatibilities with existing applications to WMF 3.0. You should not install WMF 4.0 on servers that are running the following applications:

* System Center 2012 Configuration Manager (not including SP1)
* System Center Virtual Machine Manager 2008 R2 (including SP1)
* Microsoft Exchange Server 2013, Microsoft Exchange Server 2010, and Microsoft Exchange Server 2007
* Microsoft SharePoint 2013 and Microsoft SharePoint 2010
* Windows Small Business Server 2011 Standard

# Feedback and Contact

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

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

# Important Links

WMF 3.0: <http://support.microsoft.com/kb/2506143>

WMF 3.0 Update: <http://support.microsoft.com/kb/2823180>