Windows Management Framework 3.0 Release Notes

# Overview

Windows Management Framework (WMF) 3.0 makes some updated management functionality from Windows Server 2012 available for installation on Windows 7 SP1, Windows Server 2008 R2 SP1, and Windows Server 2008 SP2. WMF 3.0 contains Windows Server 2012 versions of the following features:

* Windows PowerShell 3.0
* Windows Remote Management (WinRM) 3.0
* Windows Management Instrumentation (WMI)
* Management OData IIS Extension
* Server Manager CIM Provider

# Supported Platforms

WMF 3.0 can be installed on the following Windows configurations.

|  |  |  |
| --- | --- | --- |
| Operating System | Service Pack Level | SKUs |
| Windows 7 | Service Pack 1 | All |
| Windows Server 2008 R2  (Including Server Core) | Service Pack 1 | All except IA64 |
| Windows Server 2008  (not including Server Core) | Service Pack 2 | All except IA64 |

Please note that Windows Vista, Windows Server 2003, and Windows XP are not supported.

# Supported Languages

WMF 3.0 is available for all supported versions of Windows for the following languages:

* English
* Chinese (Simplified)
* Chinese (Traditional)
* French
* German
* Italian
* Japanese
* Korean
* Portuguese (Brazil)
* Russian
* Spanish

# System requirements

IMPORTANT: If you have installed any previous releases of Windows Management Framework 3.0 (CTP1, CTP2, Beta, or RC), you must uninstall them before installing Windows Management Framework 3.0.

WMF 3.0 requires the following prerequisite features to be installed prior to installing WMF 3.0.

* **Microsoft .NET Framework 4.0.** You can install Microsoft .NET Framework 4.0 at <http://go.microsoft.com/fwlink/?LinkID=212547>.
* **Windows 7 Service Pack 1 (SP1)** on computers running Windows 7. To install SP1, go to <http://windows.microsoft.com/installwindows7sp1>.
* **Windows Server 2008 R2 Service Pack 1 (SP1)** on computers running Windows Server 2008 R2. To install SP1, go to <http://www.microsoft.com/en-us/download/details.aspx?id=5842>
* **Windows Server 2008 Service Pack 2 (SP2)** on computers running Windows Server 2008. To install SP2, go to <http://www.microsoft.com/en-us/download/details.aspx?id=16468>

In addition, the following requirements apply:

* To install Windows PowerShell Integrated Scripting Environment (ISE) for Windows PowerShell 3.0 on computers running Windows Server 2008 R2 with Service Pack 1, before installing WMF 3.0, use Server Manager to add the optional Windows PowerShell ISE feature.
* Install the latest Windows updates before installing WMF 3.0.

# Installing WMF 3.0

IMPORTANT: If you have installed any previous releases of Windows Management Framework 3.0 (CTP1, CTP2, Beta, or RC), you must uninstall them before installing Windows Management Framework 3.0.

To install WMF 3.0:

1. Close all Windows PowerShell windows.
2. Uninstall any pre-release versions of WMF 3.0, if installed.
3. Download and run the correct package for your operating system and architecture.

|  |  |
| --- | --- |
| **Operating System / Architecture** | **Package name** |
| Windows 7 SP1 or Windows Server 2008 R2 SP1 / 64-bit | Windows6.1-KB2506143-x64.msu |
| Windows 7 SP1 / 32-bit | Windows6.1-KB2506143-x86.msu |
| Windows Server 2008 SP2 / 64-bit | Windows6.0-KB2506146-x64.msu |
| Windows Server 2008 SP2 / 32-bit | Windows6.0-KB2506146-x86.msu |

# Troubleshooting WMF 3.0 installation

Use the following techniques to resolve issues with the installation of WMF 3.0.

* **This update does not apply**.

On computers running Windows 7 and Windows Server 2008 R2, WMF 3.0 requires Service Pack 1 (SP1). If you receive a "This update does not apply" message when trying to install, verify that SP1 is installed. See the System Requirements section for links to download SP1.

WMF 3.0 also requires Microsoft .NET Framework 4.0 Full Profile. If you receive a “This update does not apply” message when trying to install, verify in Programs and Features in Control Panel that the Full Profile is installed as opposed to the Client Profile. To install Microsoft .NET Framework 4.0, go to <http://go.microsoft.com/fwlink/?LinkID=212547>

* **Windows PowerShell ISE is not installed.**

On Windows Server 2008 R2 with Service Pack 1, Windows PowerShell Integrated Scripting Environment (ISE) is installed only if the optional Windows PowerShell ISE feature is already added. Use Server Manager to see if Windows PowerShell ISE is installed or to add it if it is not installed.

# Uninstalling WMF 3.0

To uninstall any previous installations of WMF 3.0

* Use Windows Update in Control Panel to locate and uninstall the following updates:
  + KB2506143 – for Windows 7 SP1 and Windows Server 2008 R2 SP1
  + KB2506146 – for Windows Server 2008 SP2

# Getting the Latest Help Content

The Updatable Help system, introduced in Windows PowerShell 3.0, is designed to provide you with the most up-to-date help content on your local computer. Updatable Help makes it easy to download and install the newest help files and to display them at the command line, assuring that your local help topics never becomes obsolete.

Updatable Help also allows you to download the help files to a file system directory or file share to support enterprise installations and computers that do not have Internet access.

All module authors can support Updatable Help for the help files that describe the cmdlets, functions, providers, and concepts in their modules. For more information, see “Supporting Updatable Help” in MSDN.

Windows PowerShell 3.0 does not come with help files pre-installed. To download and update help files

1. Start Windows PowerShell 3.0 with the "Run as administrator" option.
2. At the command prompt, type:   
   Update-Help

The help files are available immediately. You do not need to restart Windows PowerShell.

To read the help files, use the Get-Help cmdlet. Without help files, Get-Help displays auto-generated help for commands and prompts you to use the Update-Help cmdlet to install or update the help files for your modules.

At the time of release, not all Windows modules have updatable help published online yet. When attempting to update help for modules that aren't available yet, you might see one of the following errors:

ERROR: The value of the HelpInfoUri key in the module manifest must resolve to a container or root URL on a website where the help files are stored.

ERROR: For security reasons DTD is prohibited in this XML document. To enable DTD processing set the DtdProcessing property on XmlReaderSettings to Parse and pass the settings into XmlReader.Create method.

# Backwards Compatibility with Windows PowerShell 2.0

Windows PowerShell 3.0 is designed to be backwards compatible with Windows PowerShell 2.0. Cmdlets, providers, snap-ins, modules, and scripts written for Windows PowerShell 2.0 run unchanged in Windows PowerShell 3.0.

However, due to a change in the runtime activation policy in Microsoft .NET Framework 4.0, Windows PowerShell host programs that were written for Windows PowerShell 2.0 and compiled with Common Language Runtime (CLR) 2.0 cannot run without modification in Windows PowerShell 3.0, which is compiled with CLR 4.0.

Additional changed items include:

* Additions to the Windows PowerShell language
* Improved error detection
* Changes in cmdlet and parameter behavior

To maintain backward compatibility with commands and host programs that are affected by these changes, the Windows PowerShell 2.0 and 3.0 engines are designed to run side-by-side. Also, the Windows PowerShell 2.0 engine is included in Windows 8, Windows Server 2012, and Windows Management Framework 3.0.

The Windows PowerShell 2.0 engine is intended to be used only when an existing script or host program cannot run because it is incompatible with Windows PowerShell 3.0 or Microsoft .NET Framework 4.0. Such cases are expected to be rare.

The Windows PowerShell 2.0 engine comprises PSPluginWkr.dll in $pshome and v2.0.50727 of the Windows PowerShell assemblies in the .NET Framework 2.0 Global Assembly Cache (GAC) which includes the following:

* Microsoft.PowerShell.Commands.Diagnostics.dll
* Microsoft.PowerShell.Commands.Management.dll
* Microsoft.PowerShell.Commands.Utility.dll
* Microsoft.PowerShell.ConsoleHost.dll
* Microsoft.PowerShell.Security.dll
* Microsoft.WSMan.Management.dll
* System.Management.Automation.dll

To start the Windows PowerShell 2.0 engine, use the **Version** parameter of Powershell.exe with a value of **2**.

Running the Windows PowerShell 2.0 engine does not replicate the complete Windows PowerShell 2.0 experience.

* The interactive features of Windows PowerShell 2.0, including format files, help content, the **Out-GridView** cmdlet, and Windows PowerShell ISE, are replaced by their Windows PowerShell 3.0 versions.
* The **NoProfile** parameter of PowerShell.exe cannot be specified when the value of the Version parameter is 2.0.
* The **Enable-PSRemoting** and **Disable-PSRemoting** cmdlets appear to run in Windows PowerShell 2.0, but they do not correctly configure Windows PowerShell remoting. Run these cmdlets only in Windows PowerShell 3.0.

## Changes to the Windows PowerShell Language

The Windows PowerShell language has changed in Windows PowerShell 3.0. The following changes might affect scripts written in Windows PowerShell 2.0.

| Change | Sample | Error Message | Workaround |
| --- | --- | --- | --- |
| Return from Finally is not supported | try {} finally { return } try {} finally { return 1 } try {} finally {  if ($x) {  return 1  } } | + try{}finally{return} + ~~~~~~ Control cannot leave a finally block.C3 | When return from finally is unconditional, remove the Return statement from the Try-Catch-Finally block.  try {} finally {  if ($x) {  $retval = 1  $hasretval = $true  } } if ($hasretval) {  return $retval } |
| System.Management.Automation is no longer the currently executing assembly for scripts | # From PSCX [type]::GetType("System.Management.Automation.TypeAccelerators")  # From internal test tool [Reflection.Assembly]::GetExecutingAssembly() | Various errors or incorrect behavior, depending on the use. | Depends on the particular use.  For the PSCX sample, specifying the assembly name works. |
| Read/Modify/Write operators no longer use dynamic scoping for the Read operation. Also, compound equality operators (including +=, -=, \*=, %=, ++, --) do not use dynamic scoping. The variable is always in the current scope. | $x = 1 & { $x += 1; $x } # Returns 2 in Windows PowerShell 2.0  # Returns 1 in Windows PowerShell 3.0 | No error. Results differ. | Rewrite to use only the simple assignment operator: $x = 1 & { $x = $x + 1; $x} |
| Comparing a number to a string behaves differently | 1 -eq "1.1"  # $true in Windows PowerShell 2.0; $false in Windows PowerShell 3.0  1024 -eq "1kb"  # $false in Windows PowerShell 2.0; $true in Windows PowerShell 3.0. | Depends on the use. Typically, no error but results differ. | Do not compare strings to numbers. If you must, cast explicitly.  1 -eq [int]"1.1" |
| Compile fails on C# code that uses PSObject and doesn't reference System.Core.dll | PSObject obj = val as PSObject; | "You must add a reference to System.Core.dll" | Add the reference. |
| Add-Member and PSTypeNames now apply to the underlying base object, not to PSObject. | # Run this twice  $x = @{} $x | Add-Member –MemberType NoteProperty -Name Test -Value 42  $x.Test  #In Windows PowerShell 2.0, returns $null because $x does not have the Test property. Must use the PassThru parameter to add the property to $x.  #In Windows PowerShell 3.0, returns 42. | Add-Member : Cannot add a member with the name "Foo" because a member with that name already exists. To overwrite the member, use the Force parameter. At line:1 char:11 + $y = $x | Add-Member NoteProperty Foo 42 –PassThru + ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ | Possible resolutions:  -- Add the Force parameter.  -- Remove the Add-Member call.  -- Creating a new object (by using Clone()).. |
| Using the redirection operators as arguments to some commands generates an error | # Use the new redirection tokens as a command argument echo 1> 1>> 3> 3>> 4> 4>> 5> 5>> 6> 6>> | PS> echo 1> At line:1 char:8 + echo 1> + ~ Missing file specification after redirection operator. | None |
| You cannot set the IsFilter property on an unnamed script block. | $sb = {$\_\*5} $sb.IsFilter = $true $sb2.IsFilter | PS E:\> $sb.IsFilter = $true Operation is not valid due to the current state of the object. At line:1 char:1 + $sb.IsFilter = $true + ~~~~~~~~~~~~~~~~~~~~  + CategoryInfo : InvalidOperation: (:) [], RuntimeException  + FullyQualifiedErrorId : InvalidOperation | Use a named script block, such as {process {$\_\*5}} |
| The iteration variable in a foreach statement does not have its own scope. | [int[]]$n = 1,2,3 foreach ($n in 'z') {} | Cannot convert value "z" to type "System.Int32[]". Error: "Cannot convert value "z" to type "System.Int32". Error: "Input string was not in a correct format."" | Rename the loop iteration variable, or remove the type constraint on the foreach statement. |
| You can access the properties of underlying objects in an array.  In Windows PowerShell 2.0, because the array does not have the property, no values are returned. | # Access properties of underlying objects $p = Get-Process $p.Name  # Windows PowerShell 2.0 returns $null.  # Windows PowerShell 3.0 returns process names | No error | None |
| You can index into position 0 of a scalar variable  In Windows PowerShell 2.0, this action generates an error. | # Index into scalar $a = 1 $a[0]  # Windows PowerShell 2.0 generates an error.  # Windows PowerShell 3.0 returns 1. | Unable to index into an object of type System.Int32.  At line:1 char:4  + $a[ <<<< 0]  + CategoryInfo : InvalidOperation: (0:Int32) [], RuntimeException  + FullyQualifiedErrorId : CannotIndex |  |
| Multiple #requires -  The Version parameter is not valid when the version numbers differ | #requires -version 2.0 #requires -version 3.0 | At D:\work\req.ps1:2 char:11 + #requires -version 3.0 + ~~~~~~~~ Cannot bind parameter because parameter 'version' is specified more than once. To provide multiple values to parameters that can accept multiple values, use the array syntax. For example, "-parameter value1,value2,value3". | Remove the #requires lines with the lowest version numbers. |
| Script blocks executed as delegates run in their own scope | Add-Type @" public class Invoker {  public static void Invoke(System.Action<int> func)  {  func(1);  } } "@ $a = 0 [Invoker]::Invoke({$a = 1}) $a | Returns 1 in Windows PowerShell 2.0 Returns 0 in Windows PowerShell 3.0 |  |

## Improved ERROR Detection

The following changes to error handling in Windows PowerShell 3.0 might affect scripts written in Windows PowerShell 2.0.

| Change | Sample | Error Message | Workaround |
| --- | --- | --- | --- |
| An expandable string with empty-braced variable name is now an error. | "<${}>" # expands to: <$> | + "${}" + ~ Empty ${} variable reference, there should be a name between the braces. | Use a variable name |
| An expandable string with empty sub-expression now results in different output. | "<$()>" # expands to: <$> | No error message, result of sample is: <> | Just use a '$' instead of $(). |
| In an expandable string, an incomplete variable name is an error. | $server = "msp42" $status = "online" "$server: $status" # output in V2: : online | + "$server: $status" + ~~~~~~~~ Invalid variable reference. '$' was not followed by a valid variable name character. Consider using ${} to delimit the name. | "${server}: $status" |
| A parameter followed by the comma operator is treated as unary comma operator, which is not permitted when invoking a command. | echo -InputObject ,1 | + echo -InputObject ,1 + ~ Missing argument in parameter list. | In most cases, the comma operator is a typographical error. |
| Incorrectly typed array indices raise an exception. | "abc"['def'] | Cannot convert value "def" to type "System.Int32". Error: "Input string was not in a correct format." At line:1 char:1 + "abc"['def'] + ~~~~~~~~~~~~ | Correct the script. |
| [ref] must be the final type in a type conversion sequence | [object][ref] $x = 1 | + [object][ref] $x = 1 + ~~~~~ Cannot use [ref] with other types in a type constraint. | [ref] can't be used this way, instead write:  [ref][object] $x = 1 |
| Constant folding is not performed during parsing. | 1/0 | No parsing. The error is detected at runtime. | No workaround. |
| Invalid type in a trap statement causes a runtime exception, even when the trap is not hit. | & {trap [foo]{}} | Unable to find type [foo]: make sure that the assembly containing this type is loaded. | Do not catch non-existent types |
| Inline comments in strings generate an error | $string = @" part of the string $(# comment) "@ | At D:\work\a.ps1:3 char:14 + $(# comment) + ~ Missing closing ')' in subexpression. | Do not place comments in code. |
| InitialSessionState throws an exception if a referenced assembly is not found. | PS > $iss = [System.Management.Automation.Runspaces.InitialSessionState]::Create()  PS > $iss.Assemblies.Add("foo")  PS > $iss.ThrowOnRunspaceOpenError = $true  PS > $rs = [System.Management.Automation.Runspaces.RunspaceFactory]::CreateRunspace($iss)  PS > $rs.Open() | Exception calling "Open" with "0" argument(s): "The assembly 'foo' was not loaded because no assembly was found.  Please check the assembly name and try again."  At line:1 char:1  + $rs.Open()  + ~~~~~~~~~~  + CategoryInfo : NotSpecified: (:) [], MethodInvocationException  + FullyQualifiedErrorId : DllNotFoundException | Ensure any referenced assemblies exist |

## Cmdlet Behavior Changes

This section lists additional changes in Windows PowerShell 3.0 that might affect existing scripts.

### Get-PSSession -ComputerName

In Windows PowerShell 3.0, the **ComputerName** parameter of the **Get-PSSession** cmdlet gets all user-managed sessions that connect to the specified computer. In Windows PowerShell 2.0, the **ComputerName** parameter gets only the sessions that originated in the current session and connected to the specified computer.

In Windows PowerShell 3.0, the Disconnected Sessions feature lets you to disconnect from a session and reconnect to it later from the same or a different computer. As such, you need to be able to get all sessions, even when connecting from a different computer.

In addition, the new **State** parameter of **Get-PSSession** gets user-managed sessions that have a particular state, such as **Disconnected**.

To get the Windows PowerShell 2.0 behavior of the **ComputerName** parameter, run a **Get-PSSession** command without the **ComputerName** parameter. This gets all sessions that originated in the current session. Then, use the **Select-Object** or **Where-Object** cmdlets to get sessions that are connected to particular computers.

**Examples:**

* To get all user-managed sessions that connect to the Server01 computer, type:

Get-PSSession –ComputerName Server01

* To get user-managed sessions that were created in the current session and connect to the Server01 computer, type:

Get-PSSession | where {$\_.ComputerName -eq "Server01"}

### Receive-Job -Location

In Windows PowerShell 3.0, the **ValueFromPipelineByPropertyName** attribute has been removed from the **Location** parameter of the **Receive-Job** cmdlet.

When you use **AsJob** parameter of the **Invoke-Command** cmdlet, the value of the **Location** property of the resulting job object is a concatenation of the location values of all child jobs. When you run a command on more than one computer, the value of the **Location** property is a comma-separated list of all computers on which the child jobs ran. As a result, when you pipe the job object to the **Receive-Job** cmdlet, the command fails, because the value of the **Location** property is not a valid computer name.

Because the **ReceiveJobCommand** class is public, we cannot change the value of the **Location** property. Instead, we removed the **ValueFromPipelineByPropertyName** attribute of the **Location** property, so that **Receive-Job** commands do not fail.

This breaking change affects only commands in which **Location** is the only property that is bound from the pipeline. For example: $ObjectWithLocationProperty | Receive-Job -Job (Get-Job).

### Windows PowerShell Core Snap-ins Have Been Converted to Modules

In Windows PowerShell 1.0 and 2.0, the core Windows PowerShell cmdlets and providers are implemented as snap-ins. These snap-ins are automatically loaded into every session. Users cannot remove them.

In Windows PowerShell 3.0, the packages that contain the core cmdlets and providers have been converted to modules. Users can add and remove the core modules. The exception is **Microsoft.PowerShell.Core**, which remains a snap-in.

The new modules have the same names as the old snap-ins, as follows:

* Microsoft.PowerShell.Diagnostics
* Microsoft.PowerShell.Host
* Microsoft.PowerShell.Management
* Microsoft.PowerShell.Security
* Microsoft.PowerShell.Utility
* Microsoft.WSMan.Management

In Windows PowerShell 3.0, to get the Core modules, use the **Get-Module** cmdlet. The **Get-PSSnapIn** cmdlet returns only **Microsoft.PowerShell.Core**. Also, the **PSSnapIn** property of Windows PowerShell core cmdlets is populated only for cmdlets in the **Microsoft.PowerShell.Core** snap-in. The **ModuleName** property is populated for cmdlets that are packaged in modules and snap-ins.

The core snap-ins are converted to modules by the use the of [**CreateDefault2**](http://msdn.microsoft.com/en-us/library/windows/desktop/system.management.automation.runspaces.initialsessionstate.createdefault2(v=vs.85).aspx) method of the **InitialSessionState** class. As such, the core snap-ins appear as modules only in host programs, such as the Windows PowerShell console and Windows PowerShell ISE, that use the **CreateDefault2** method to create initial session states. In remote sessions and in older host programs that do not use **CreateDefault2**, the core packages still appear as snap-ins.

### CompilerParameters and ReferencedAssemblies parameters on Add-Type are now Mutually Exclusive

Beginning in Windows PowerShell 3.0, the **ReferencedAssemblies** and **CompilerParameters** parameter of the **Add-Type** cmdlet are exclusive. They cannot be specified in the same command.

In Windows PowerShell 2.0, when an **Add-Type** command includes both the **ReferencedAssemblies** and **CompilerParameters** parameters, the value of the **ReferencedAssemblies** parameter is ignored without notice. This practice might result in an error.

The value of the **CompilerParameters** parameter can include compiler parameters and referenced assemblies. To specify both, use only the **CompilerParameter** parameter.

### Line Breaks in XML Attributes are Ignored

To improve security, the Windows PowerShell 3.0 command parser enforces XML standards more rigorously. Beginning in Windows PowerShell 3.0, line breaks in XML attributes are ignored.

To include a line break in an XML attribute, use the XML newline special character code (**&#10;** ) or use System.Xml.XmlDocument.LoadXml().

For example:

PS C:\> [xml]$x = '<Element Attribute="line

break" />'

# Windows PowerShell 2.0

PS C:\> $x.element.attribute

line

break

# Windows PowerShell 3.0

PS C:\> $x.element.attribute

line break

PS C:\> [xml]$x = '<Element Attribute="line**&#10;**break" />'

PS C:\> $x.element.attribute

line

break

### Select-Object Optimization Does Not process the Entire Collection

In Windows PowerShell 2.0, the **Select-Object** cmdlet processed all items in a collection, even when the pipeline included a Select-Object command that selected only the first few items in the collection.

For example, given the following command, the **Get-WinEvent** cmdlet gets all event logs, even when the user selects only the first five event logs.

Get-WinEvent | Select-Object -First 5

In Windows PowerShell 3.0, to improve command execution time, especially when working with large collections, the **Select-Object** cmdlet now returns after it has processed the number of items specified by the **First** parameter of **Select-Object**. For example, given the previous command, the **Get-WinEvent** cmdlet returns only the first five event logs.

Scripts that expect Windows PowerShell to process an entire collection will get unexpected results.

For example, in Windows PowerShell 2.0, the following command replaces the file name extension of all .gif files with the .jpg file name extension and then gets the first five files. In Windows PowerShell 3.0, the command replaces the file name extension of only the first five .gif files with the .jpg file name extension, and then gets those five files.

dir \*.gif | foreach {$\_ -replace ".gif", ".jpg"} | Select-Object -First 5

To get the un-optimized Windows PowerShell 2.0 behavior, use the **Wait** parameter of **Select-Object**.

dir \*.gif | foreach {$\_ -replace ".gif", ".jpg"} | Select-Object -First 5 -Wait

### WinRM Put Operation Ignores Missing Properties

In WinRM 2.0, when a WinRM service PUT request for a resource is missing an optional property, the value of the missing property is set to NULL ($null). In WinRM 3.0, when a WinRM service PUT request for a resource is missing an optional property, the property is ignored. Any property values that are set on the server remain unchanged.

# Known Issues

## SharePoint Server 2010 Management Shell Stops Working After Installing Windows Management Framework 3.0

After installing Windows Management Framework 3.0, the SharePoint Management Shell returns the following error:

“Microsoft SharePoint is not supported with version 4.0.30319.239 of the Microsoft .Net Runtime.”

The SharePoint 2010 Management Shell is not supported on Windows PowerShell 3.0.

To resolve the problem:

1. Use the following command to start the Windows PowerShell 2.0 engine.

powershell.exe -Version 2.0

1. Use the following command to add the SharePoint snap-in

Add-PSSnapin Microsoft.SharePoint.PowerShell

## Cannot Create Remote Sessions to Computers with WMF CTP1 or CTP2

Computers running Windows Management Framework 3.0 cannot create remote sessions ("PSSessions”) on computers running Windows Management Framework 3.0 CTP1 or CTP2.

However, computers running Windows Management Framework 3.0 can create PSSessions on computers running Windows Management Framework 3.0 RTM, RC, Beta, or Windows PowerShell 2.0.

Windows Management Framework 3.0 includes version 2.2 of the Windows PowerShell remoting protocol. Version **2.2** of the protocol is not compatible with version **2.10X**, which was included in Windows Management Framework 3.0 CTP1 and CTP2.

To resolve this problem, install the final version of Windows Management Framework 3.0 on all affected computers.

## Remoting is Disabled After Installing WMF 3.0

After installing Windows Management Framework 3.0 on computers running Windows Server 2008 SP2, Windows PowerShell remoting is disabled, even if it was previously enabled. To enable Windows PowerShell remoting, start Windows PowerShell with the "Run as administrator" option, and then use the Enable-PSRemoting cmdlet.

## Server Manager cannot get performance data for DOWNLEVEL computers

Windows Management Framework 3.0 includes a CIM provider that enables you to use Server Manager in Windows Server 2012 to manage computers running Windows Server 2008 R2 SP1 and Windows Server 2008 SP2.However, when using Windows Management Framework 3.0, the Server Manager Roles page cannot collect performance counters from computers running Windows Server 2008 R2 SP1 or Windows Server 2008 SP2.To enable the Roles page to collect performance data, install KB2682011. For more information, see <http://support.microsoft.com/kb/2682011>.