

Operations Manager 2007 R2 Management Pack Module Reference

Microsoft Corporation

Feedback

Send suggestions and comments about this document to [staceyf@microsoft.com](mailto:staceyf@microsoft.com?subject=Module%20Reference%20published%20December%202008).

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows Server, Windows Vista, and Active Directory are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are property of their respective owners.

Revision History

|  |  |
| --- | --- |
| Release Date | Changes |
| December 19, 2008 | * First release of this document. |
| June 19, 2009 | R2 updates. SNMP and Windows Powershell module documentation. |

Contents

[Operations Manager 2007 R2 Management Pack Module Reference 5](#_Toc234165666)

[Microsoft.SystemCenter.CollectEvent 6](#_Toc234165667)

[Microsoft.SystemCenter.DataWarehouse.PublishEventData 8](#_Toc234165668)

[Microsoft.SystemCenter.WSManagement.Getter 11](#_Toc234165669)

[Microsoft.Windows.EventProvider 19](#_Toc234165670)

[Microsoft.Windows.ExistingWin32ServiceInformationProbe 26](#_Toc234165671)

[Microsoft.Windows.RegistryProbe 31](#_Toc234165672)

[Microsoft.Windows.ScriptProbeAction 37](#_Toc234165673)

[Microsoft.Windows.Win32ServiceInformationProbe 46](#_Toc234165674)

[Microsoft.Windows.WmiProbe 52](#_Toc234165675)

[System.ApplicationLog.GenericCSVLog.FilteredEventDataProvider 57](#_Toc234165676)

[System.ApplicationLog.GenericLog.FilteredEventProvider 64](#_Toc234165677)

[System.CommandExecuterProbe 72](#_Toc234165678)

[System.ConsolidatorCondition 83](#_Toc234165679)

[System.Discovery.ClassSnapshotDataMapper 100](#_Toc234165680)

[System.Event.GenericDataMapper 108](#_Toc234165681)

[System.ExpressionFilter 119](#_Toc234165682)

[System.Health.GenerateAlert 131](#_Toc234165683)

[System.Performance.AveragerCondition 138](#_Toc234165684)

[System.Performance.DataGenericMapper 141](#_Toc234165685)

[System.Performance.OptimizedDataProvider 146](#_Toc234165686)

[System.Scheduler 153](#_Toc234165687)

[System.SnmpProbe 160](#_Toc234165688)

[System.SnmpEventProvider 175](#_Toc234165689)

[System.SnmpTrap.EventProvider 178](#_Toc234165690)

[System.SnmpTrapProvider 180](#_Toc234165691)

[System.SnmpTrapEventProvider 183](#_Toc234165692)

[System.SnmpTrap.FilteredEventProvider 186](#_Toc234165693)

[System.SnmpScanQuery.EventProvider 190](#_Toc234165694)

[System.SnmpQueryProvider 193](#_Toc234165695)

[System.SnmpQuery.FilteredEventProvider 197](#_Toc234165696)

[System.SnmpScanProbe 201](#_Toc234165697)

[System.SnmpScanQueryProvider 204](#_Toc234165698)

[System.SnmpScanQuery.FilteredEventProvider 207](#_Toc234165699)

[Appendix: PowerShell Modules 211](#_Toc234165700)

[Microsoft.Windows.TimedPowerShell.DiscoveryProvider 219](#_Toc234165701)

[Microsoft.Windows.PowerShellPropertyBagProbe 223](#_Toc234165702)

[Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe 231](#_Toc234165703)

[Microsoft.Windows.PowerShellDiscoveryProbe 234](#_Toc234165704)

Operations Manager 2007 R2 Management Pack Module Reference

Modules are the main building blocks of a management pack. Modules are composed together in various combinations to create workflows, which are the foundation of Operations Manager. This reference guide documents the most commonly used management pack modules.

Modules

This reference covers the following commonly used modules:

[Microsoft.SystemCenter.CollectEvent](#zf7ae4a7fab534531abfe2986478f4dac)

[Microsoft.SystemCenter.DataWarehouse.PublishEventData](#z6508a3426ccf494cbd951fcf10feb963)

[Microsoft.SystemCenter.WSManagement.Getter](#z68dd8363c38c4262a03d978521fd19db)

[Microsoft.Windows.EventProvider](#zdc2cb385881a4ef6947fbbb60f1553ad)

[Microsoft.Windows.ExistingWin32ServiceInformationProbe](#z90b8f97bb7ee46e6a150feb6cd4ad906)

[Microsoft.Windows.RegistryProbe](#zddd0c74861c74d88b7925909edda994a)

[Microsoft.Windows.ScriptProbeAction](#z7d0999603afd4d1889a2b779c9b5c7de)

[Microsoft.Windows.Win32ServiceInformationProbe](#z8d77519933a649beb41fa3929e249f1a)

[Microsoft.Windows.WmiProbe](#zea3d6a47757c46768448247a33a745cb)

[System.ApplicationLog.GenericCSVLog.FilteredEventDataProvider](#ze84bedc749264da49a1c57ff939f0d7f)

[System.ApplicationLog.GenericLog.FilteredEventProvider](#z27a5acba882b4dffaedcb36a01cb2e61)

[System.CommandExecuterProbe](#z84fa12e7ac0f45b5bceb6f960db939f1)

[System.ConsolidatorCondition](#zf2c6fbb51cdd4c46860f810a117bf29d)

[System.Discovery.ClassSnapshotDataMapper](#z5f2b542c5b0a45ec949acaf92abcb69d)

[System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298)

[System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca)

[System.Health.GenerateAlert](#zbe1fbde31d2b466096c2eee37d1d0be7)

[System.Performance.AveragerCondition](#z9a75f2f9658b4848bdce99cafb77bf77)

[System.Performance.DataGenericMapper](#zf040319cbc3641ef9639f07aff38e98e)

[System.Performance.OptimizedDataProvider](#z99cabef2e92b49cd96ca861cb3d2c0b5)

[System.SnmpProbe](#za9cf835b51ab4780955b1a384d2ae6f7)

[System.SnmpEventProvider](#z1b2b9e94d10841e7982beee426f526d0)

[System.SnmpTrap.EventProvider](#z7686beeb67b145ce89ee8c56e80ac64a)

[System.SnmpTrapProvider](#zcbf30137a5e54a8b999cbf97c9685f50)

[System.SnmpTrapEventProvider](#z22acfcb9ade14dd58fb27e63faca47db)

[System.SnmpTrap.FilteredEventProvider](#z67ea6fc6e6e54dbab04e33a91fe84d6a)

[System.SnmpScanQuery.EventProvider](#z89384c8f1c3f40d3bf58646e1d9d2f5e)

[System.SnmpQueryProvider](#z704961a8e7754ef6b65bc748503b1636)

[System.SnmpQuery.FilteredEventProvider](#za02c3dabd5794b92a1212526bef75a33)

[System.SnmpScanProbe](#z7f57720b6f0e40839c188942854dbdbe)

[System.SnmpScanQueryProvider](#zf35161d4a0d94fbb9498016808a28c16)

[System.SnmpScanQuery.FilteredEventProvider](#zca85ace150eb4dc58a18a7426112e9e0)

[Microsoft.Windows.TimedPowerShell.DiscoveryProvider](#z3a8388cf182844d492bd67cf715f40cc)

[Microsoft.Windows.PowerShellPropertyBagProbe](#z62e9c3a7f5fa49aaac8dba8da7b3a850)

[Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe](#zaa5d383a070a479da544600f119e1d12)

[Microsoft.Windows.PowerShellDiscoveryProbe](#z648d3e8e54874d7fa0baab05b11e070a)

[System.Scheduler](#z8d232c0e53514b4c900480d34ddf5a6a)

Microsoft.SystemCenter.CollectEvent

The Microsoft.SystemCenter.CollectEvent write action module type takes Operations Manager event data (System.Event.Data) as input and writes the event to the Operations Manager database.

Parameters

This module takes no configuration parameters.

Remarks

This module is commonly used by any rule that generates event data that must be stored in the Operations Manager database. It is often used together with the [Microsoft.SystemCenter.DataWarehouse.PublishEventData](#z6508a3426ccf494cbd951fcf10feb963) write action module.

To store non-event data in the Operations Manager database, using this module, you must first map the data to the event format by using the [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) mapper module that is defined in the System.Library management pack.

Examples

The following code example is a rule that collects an event from the Windows event log and writes it to the Operations Manager database only:

<Rule ID="Microsoft.Samples.Collect101" Target="Microsoft.Samples.ApplicationX" Enabled="true">

<Category>AvailabilityHealth</Category>

<DataSources>

<DataSource ID="DS" TypeID="Windows!Microsoft.Windows.EventProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<LogName>Application</LogName>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">101</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

</WriteActions>

</Rule>

The following WriteAction element writes the event to both the data warehouse and the Operations Manager database:

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishEventData"/>

</WriteActions>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| [Microsoft.SystemCenter.DataWarehouse.PublishEventData](#z6508a3426ccf494cbd951fcf10feb963) | Microsoft.SystemCenter.Library | Writes event data to the data warehouse. |
| [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) | System.Library | Maps any data type to event data. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.SystemCenter.DataWarehouse.PublishEventData

The Microsoft.SystemCenter.DataWarehouse.PublishEventData write action module type takes Operations Manager event data (System.Event.Data) as input and writes the event to the System Center Data Warehouse database.

Parameters

This module type takes no configuration parameters.

Remarks

This module is used when you want to store data in the System Center Data Warehouse database, and it is used for any rule that generates event data. It is often used together with the Microsoft.SystemCenter.CollectEvent write action module type, which writes data to the Operations Manager database.

To store data in the System Center data warehouse database that is not of type event using this module, you must first map the data to the event format by using the [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) mapper module that is defined in the System.Library management pack.

Examples

The following code example is a rule that collects an event from the Windows event log and writes it to the System Center data warehouse database only:

<Rule ID="AuthorMPs.ModuleSamples.ApplicationX.Collect101" Target="AuthorMPs.ModuleSamples.ApplicationX" Enabled="true">

<Category>AvailabilityHealth</Category>

<DataSources>

<DataSource ID="DS" TypeID="Windows!Microsoft.Windows.EventProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<LogName>Application</LogName>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">101</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishEventData"/>

</WriteActions>

</Rule>

The following WriteAction element writes the event to both the data warehouse and the Operations Manager database:

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishEventData"/>

</WriteActions>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| [Microsoft.SystemCenter.CollectEvent](#zf7ae4a7fab534531abfe2986478f4dac) | Windows.Library | Writes event data to the Operations Manager database. |
| [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) | System.Library | Maps any data type to event data. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.SystemCenter.WSManagement.Getter

The Microsoft.SystemCenter.WSManagement.Getter module is a probe action that calls the WS-Management Get operation to access information about a WS-Management object or objects. This information can be useful for either discovery or monitoring purposes.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Get | Microsoft.SystemCenter. WSManagement.Schemas/GetSchema | Contains the schema used to specify the set of properties required to perform a WS-Management Get operation |

The Microsoft.SystemCenter.WSManagementSchemas/GetSchema has the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Protocol | String | Contains a string specifying the ProtocolType that is used to communicate between computers. The supported values are HTTP and HTTPS. |
| TargetSystem | String | Specifies the system against which the request is submitted. This can be a fully qualified domain name or an IP address. For more information, see notes later in this document. |
| Port | Integer | Specifies the port number for the request. Must be a value in the range of 0 to 65535. |
| Authentication | String | Contains the AuthenticationType to use for communicating. Currently supported values are:   None   Basic   Kerberos   Digest   Negotiate |
| UserName | String | Contains the user name used for authentication. For use with Basic, Digest, and Negotiate authentication only. For more information, see notes later in this document. |
| Password | String | Contains the password that is used for authentication. For use with Basic, Digest, and Negotiate authentication only. For more information, see notes later in this document. |
| SkipCACheck | Boolean | Specifies whether to skip any validations against the certification authority (CA) on the target system. Enabling this flag causes the modules to trust the certificate on a target system, even if they are self-signed. |
| SkipCNCheck | Boolean | Specifies that machine name and server certificate common name (CN) check should be skipped. For more information, see notes later in this document. |
| OutputErrorIfAny | Boolean | Optional. Specifies whether to output any errors. Default value is false. |
| UTF | String | Specifies the encoding format. Currently supported values are UTF-8 and UTF-16. |
| Uri | String | Contains the complete URI for the objects being queried for. |
| Selector | String | Contains the selector string used to filter and select objects being queried for. |
| TimeOutInMS | unsigned Integer | Specifies the amount of time to wait for the response from the computer queried. Default value is 60,000 milliseconds (60 seconds). For more information, see notes later in this document. |

TargetSystem

This parameter is often passed by a substitution parameter based on the target or other configuration. For example:

$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$

UserName and Password

For Kerberos and Negotiate authentication, the UserName parameter can still be passed, but it does not affect the outcome of the calling method.

You can also pass in the UserName of a RunAsAccount associated with a RunAsProfile under which WsMan executes the module. For example:

$RunAs[Name="WSMan!Microsoft.SystemCenter.WSManagement.WSManActionAccount"]/UserName$

The Password parameter can be passed in the same way. For example:

$RunAs[Name="WSMan!Microsoft.SystemCenter.WSManagement.WSManActionAccount"]/Password$

This assumes the reference to the Microsoft.SystemCenter.WSManagement.Library has been set to WSMan. For example:

<Reference Alias="WSMan">

<ID>Microsoft.SystemCenter.WSManagement.Library</ID>

<Version>6.0.6278.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

SkipCNCheck

When using HTTPS, the computer name must match the server's certificate common name (CN) unless the SkipCNCheck parameter is set to true. This parameter should be set to true when communicating with a device using HTTPS and using an IP address (rather than a fully qualified domain name), or when the name under which the certificate has been issued is different from the name of the target system.

TimeOutInMS

When this parameter is overridden, WS-Management uses the overridden value or four times NetworkDelayms, whichever is greater. The value in NetworkDelayms can be found by running the following at the command prompt:

winrm get winrm/Config/Client

Remarks

WS-Management is a standards-based Web services API described by the Distributed Management Task Force (DMTF). This probe action module type depends on the Windows implementation of this standard, called Windows Remote Management (WINRM).

To use this module, the computer that is running the workflows that calls the WS-Management modules needs to have WS-Management version 1.1 installed. This also must be installed on each Operations Manager agent or management server where rules, discoveries, monitors, or tasks that use WS-Management must execute.

WS-Management can be downloaded from the [Microsoft Download Center](http://go.microsoft.com/fwlink/?LinkId=136941).

This module is configured to use the global RunAs account Microsoft.SystemCenter.WSManagement.WSManActionAccount. The Profile Display name for this account is WS-Management Action account.

If this profile is not configured, the workflow uses the agent’s Action Account that was specified when the agent was installed.

Examples

The following example shows a Get schema and the resulting data item.

<Get>

<Protocol>http</Protocol>

<TargetSystem>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$</TargetSystem>

<Port>80</Port>

<Authentication>kerberos</Authentication>

<UserName />

<Password />

<SkipCACheck>true</SkipCACheck>

<SkipCNCheck>true</SkipCNCheck>

<UTF>utf-8</UTF>

<Uri>http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/Win32\_Service</Uri>

<Selector>Name=spooler</Selector>

<TimeOutInMS>2000</TimeOutInMS>

</Get>

The following code shows the data item output from the preceding Get schema:

<DataItem type =" Microsoft.SystemCenter.WSManagement.WSManData " time =" 2008-10-23T11:12:51.1451250-04:00 " sourceHealthServiceId =" CC9FCD47-4887-8AE7-A72F-2EACF9C2F1D9 ">

<WsManData>

<p:Win32\_Service xsi:type =" p:Win32\_Service\_Type " xml:lang =" en-US ">

<p:AcceptPause>false</p:AcceptPause>

<p:AcceptStop>true</p:AcceptStop>

<p:Caption>Print Spooler</p:Caption>

<p:CheckPoint>0</p:CheckPoint>

<p:CreationClassName>Win32\_Service</p:CreationClassName>

<p:Description>Manages all local and network print queues and controls all printing jobs. If this service is stopped, printing on the local machine will be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start.</p:Description>

<p:DesktopInteract>true</p:DesktopInteract>

<p:DisplayName>Print Spooler</p:DisplayName>

<p:ErrorControl>Normal</p:ErrorControl>

<p:ExitCode>0</p:ExitCode>

<p:InstallDate xsi:nil =" true " />

<p:Name>spooler</p:Name>

<p:PathName>C:\WINDOWS\system32\spoolsv.exe</p:PathName>

<p:ProcessId>1252</p:ProcessId>

<p:ServiceSpecificExitCode>0</p:ServiceSpecificExitCode>

<p:ServiceType>Own Process</p:ServiceType>

<p:Started>true</p:Started>

<p:StartMode>Auto</p:StartMode>

<p:StartName>LocalSystem</p:StartName>

<p:State>Running</p:State>

<p:Status>OK</p:Status>

<p:SystemCreationClassName>Win32\_ComputerSystem</p:SystemCreationClassName>

<p:SystemName>RED-DC-01</p:SystemName>

<p:TagId>0</p:TagId>

<p:WaitHint>0</p:WaitHint>

</p:Win32\_Service>

</WsManData>

</DataItem>

The following example demonstrates using this module in a management pack:

<ManagementPack ContentReadable="true" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<Manifest>

<Identity>

<ID>WSManDemo</ID>

<Version>1.0.0.0</Version>

</Identity>

<Name>WSManDemo</Name>

<References>

<Reference Alias="SC">

<ID>Microsoft.SystemCenter.Library</ID>

<Version>6.0.6278.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="Windows">

<ID>Microsoft.Windows.Library</ID>

<Version>6.0.6278.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="Health">

<ID>System.Health.Library</ID>

<Version>6.0.6278.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="System">

<ID>System.Library</ID>

<Version>6.0.6278.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="MicrosoftSystemCenterWSManagementLibrary">

<ID>Microsoft.SystemCenter.WSManagement.Library</ID>

<Version>6.0.6278.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

</References>

</Manifest>

<Monitoring>

<Tasks>

<Task ID="WSMan.Demo.Task.Get" Accessibility="Internal" Enabled="true" Target="Windows!Microsoft.Windows.Computer" Timeout="300" Remotable="true">

<Category>Custom</Category>

<ProbeAction ID="wsmanget" TypeID="MicrosoftSystemCenterWSManagementLibrary!Microsoft.SystemCenter.WSManagement.Getter">

<Get>

<Protocol>http</Protocol>

<TargetSystem>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$</TargetSystem>

<Port>80</Port>

<Authentication>kerberos</Authentication>

<UserName />

<Password />

<SkipCACheck>true</SkipCACheck>

<SkipCNCheck>true</SkipCNCheck>

<UTF>utf-8</UTF>

<Uri>http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/Win32\_Service</Uri>

<Selector>Name=spooler</Selector>

<TimeOutInMS>2000</TimeOutInMS>

</Get>

</ProbeAction>

</Task>

</Tasks>

</Monitoring>

<LanguagePacks>

<LanguagePack ID="ENU" IsDefault="true">

<DisplayStrings>

<DisplayString ElementID="WSManDemo">

<Name>WS-Man Demo</Name>

</DisplayString>

<DisplayString ElementID="WSMan.Demo.Task.Get">

<Name>WSMAn Get</Name>

<Description />

</DisplayString>

</DisplayStrings>

</LanguagePack>

</LanguagePacks>

</ManagementPack>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.SystemCenter.WSManagement.TimedGetter | Microsoft.SystemCenter.WSManagement.Library | A WS-ManagementGetter module with a timer. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

[Resource URIs](http://go.microsoft.com/fwlink/?LinkId=136942)

[Windows Remote Management Command Line Tool](http://go.microsoft.com/fwlink/?LinkId=136945)

[Windows Remote Management](http://go.microsoft.com/fwlink/?LinkId=136946)

Microsoft.Windows.EventProvider

The Microsoft.Windows.EventProvider data source module type provides filtered Windows event data of type Microsoft.Windows.EventData from any event log on a Windows-based computer.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ComputerName | ComputerNameType | Optional. Specifies the name of the computer to receive events from. |
| LogName | String | Contains the name of the Windows event log to receive events from. |
| AllowProxying | Boolean | Optional. Specifies whether the module should collect events that do not originate from the computer that is specified in the ComputerName parameter. The default value is false if this parameter is not specified. |
| Expression | ExpressionType | Contains the filter expression to apply to the events from the event log. |

No parameters are defined as overrideable in this module.

ComputerName

The ComputerName parameter is most commonly passed to the module by using the $Target notation, although it can be specified as any string. This module attempts to connect either to the local computer’s event log or to a remove event log, depending on the configuration passed.

For a workflow that is targeted at Windows Computer, the ComputerName element is specified as shown in the following code:

<ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

If the target is a class hosted by computer hosting, references are added as required. For example:

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

The ComputerName parameter is optional, but as a best practice, it should be specified.

The ComputerName parameter must be specified if the workflow that is being used is run for agentless computers or for virtual computers running on a Windows cluster. If the parameter is not specified, the module runs against the local event log of the computer that is executing the workflow; this works for agent monitoring scenarios but not for agentless monitoring scenarios.

If you omit this parameter, ensure that the rule or monitor has the remoteable attribute marked as false to prevent agentless execution, because the incorrect event log is used if the computer name is not specified. Omitting this parameter also prevents running against a virtual computer that is running on a Windows cluster.

LogName

The LogName is the name of the event log to read event from. A single instance of the module can read only from a single event log, although if required it can be combined with other modules to read and correlate between multiple event logs. An example configuration is:

<LogName>Application</LogName>

AllowProxying

By default, the event provider module reads only events that are generated by the computer that is being monitored. It is possible for another computer to log an event to the monitored computer’s event log; in this case, the event is logged using the name of the computer logging the event.

If AllowProxying is set to true, events generated from computers other than the computer being monitored are generated by the module. If AllowProxying is set to false, only locally generated events are collected.

Unless there is a specific reason to enable this for your specific monitoring scenario, it is recommended that you set AllowProxyingto false or omit this setting.

The following code is an example configuration:

<AllowProxying>True</AllowProxying>

Expression

The expression provides the filter criteria to define the events to be received and passed to the next module in the workflow. This parameter takes an ExpressionType, which is fully defined in the [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) module type documentation.

The data from the event log is of type Microsoft.Windows.EventData. The following code shows an example data item from the event log:

<DataItem type="Microsoft.Windows.EventData" time="2008-10-15T19:03:38.0000000-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<EventOriginId>{F6745F03-173A-409B-AFE2-FA00E1B8C8BE}</EventOriginId>

<PublisherId>{07662796-8DD4-4E1B-6676-B5CB4A8E4CAA}</PublisherId>

<PublisherName>HealthService</PublisherName>

<EventSourceName>HealthService</EventSourceName>

<Channel>Operations Manager</Channel>

<LoggingComputer>red-comp1.red.local</LoggingComputer>

<EventNumber>1073743034</EventNumber>

<EventCategory>1</EventCategory>

<EventLevel>4</EventLevel>

<UserName>N/A</UserName>

<RawDescription><![CDATA[New configuration became active. Management group "%1", configuration id:"%2". ]]></RawDescription>

<LCID>1033</LCID>

<Params>

<Param>red-comp1</Param>

<Param><![CDATA[62 7D 50 5E 6D A9 FD 07 3D 9B 13 6B A9 8B FE 41 94 DA 66 CE ]]></Param>

</Params>

<EventData>

<DataItem type="System.XmlData" time="2008-10-15T19:03:38.5035274-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<EventData>

<Data>red-comp1</Data>

<Data>62 7D 50 5E 6D A9 FD 07 3D 9B 13 6B A9 8B FE 41 94 DA 66 CE </Data>

</EventData>

</DataItem>

</EventData>

<EventDisplayNumber>1210</EventDisplayNumber>

<EventDescription><![CDATA[New configuration became active. Management group "stwilsonx20", configuration id:"62 7D 50 5E 6D A9 FD 07 3D 9B 13 6B A9 8B FE 41 94 DA 66 CE ". ]]></EventDescription>

<Keywords>36028797018963968</Keywords>

</DataItem>

Do not use the $Data syntax when referencing values in the data item using the expression module. For more information, see [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca).

The following are some common examples of referencing data as part of the expression:

|  |  |
| --- | --- |
| Event Parameter | XPath |
| Event Source | PublisherName |
| Event ID | EventDisplayNumber |
| Event Parameter 1 | Params/Param[1] |
| Event Parameter 2 | Params/Param[2] |

The following code shows an example of an expression that could be used as a filter to pass the data item shown in the previous example:

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">1210</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">HealthService</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

Remarks

This module type is commonly used in rules that collect events or alert on events from the Windows event logs.

Examples

The following code shows an example of an event collection rule using the Microsoft.Windows.EventProvider module type. In this example, the rule looks for an event from the EventCreate source with an event ID of 101 and with a specific instance property in parameter 1. This event is then written to the Operations database and the data warehouse.

<Rule ID="Microsoft.Samples.CollectEvent101" Target="Microsoft.Samples.ApplicationX">

<Category>AvailabilityHealth</Category>

<DataSources>

<DataSource ID="DS" TypeID="Windows!Microsoft.Windows.EventProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<LogName>Application</LogName>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">101</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">$Target/Property[Type="Microsoft.Samples.ApplicationX"]/ServiceName$</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishEventData"/>

</WriteActions>

</Rule>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.EventCollector | Microsoft.Windows.Library | Provides the same functionality as the Microsoft.Windows.EventProvider module type except it defines the AllowProxying parameter as overrideable. Use for event collection when you want to allow the user to be able to override AllowProxying. |
| Microsoft.Windows.BaseEventProviderMicrosoft.Windows.Library | Microsoft.Windows.Library | Provides the same functionality as the Microsoft.Windows.EventProvider module type but does not have an expression filter. Use when composing a new module type or monitor type. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.ExistingWin32ServiceInformationProbe

The Microsoft.Windows.Win32ServiceInformationProbe probe action module type is used to query Windows service information from the Service Control Manager. A module of this type takes a trigger as input, which means it can accept any data type to trigger execution. The module returns a System.PropertyBagData type that contains all the properties of the service. This data can be used in monitoring workflows or discovery workflows. Only data on installed services is returned by a module of this type.

Parameters

The Microsoft.Windows.Win32ServiceInformationProbe probe action module type accepts a System.PropertyBag data type with the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ComputerName | String | The name of the computer to query. |
| ServiceName | String | The service name to query. |

The ComputerName configuration is most commonly passed to the module by using the $Target notation, although it can be specified as any string.

The module attempts to connect to either the local computer’s Service Control Manager or to a remote computer’s Service Control Manager, depending on the configuration that is passed.

For a workflow that is targeted at a Windows-based computer, the ComputerName element is be specified as follows:

<ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

If the target is a class that is hosted by computer hosting, references are added as required. For example:

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

The service name specifies the short name of the Windows service to collect data on. This can be found from the properties page of the Services MMC snap-in. For example:

<ServiceName>spooler</ServiceName>

Remarks

The module outputs a property bag object that contains data about the service that was queried. The following code example shows an example data item:

<DataItem type="System.PropertyBagData" time="2008-10-08T15:52:27.0027578-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<Property Name="State" VariantType="3">4</Property>

<Property Name="ServiceType" VariantType="3">16</Property>

<Property Name="StartMode" VariantType="3">2</Property>

<Property Name="ErrorControl" VariantType="3">1</Property>

<Property Name="TagId" VariantType="3">0</Property>

<Property Name="ProcessId" VariantType="3">1880</Property>

<Property Name="Name" VariantType="8">MSSQLSERVER</Property>

<Property Name="BinaryPathName" VariantType="8">"C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\sqlservr.exe" -sMSSQLSERVER</Property>

<Property Name="LoadOrderGroup" VariantType="8"></Property>

<Property Name="Dependencies" VariantType="8"></Property>

<Property Name="ServiceStartName" VariantType="8">LocalSystem</Property>

<Property Name="DisplayName" VariantType="8">SQL Server (MSSQLSERVER)</Property>

<Property Name="Description" VariantType="8">Provides storage, processing and controlled access of data and rapid transaction processing.</Property>

</DataItem>

The data item is a property bag that contains the following properties:

|  |  |
| --- | --- |
| Property | Description |
| State | The current service status. Possible values are:   0 - Unknown   1 - Stopped   2 – Start pending   3 – Stop pending   4 - Running   5 – Continue pending   6 – Pause pending   7 - Paused   8 – Service not found   9 – Server not found |
| ServiceType | The type of service. Possible values are:   1 – Kernel/driver   2 – File system driver   16 – Own process   32 – Share process |
| StartMode | The startup type of the service. Possible values are:   0 – Boot start   1 – System start   2 - Automatic   3 - Manual   4 - Disabled |
| ErrorControl | Severity of the error and the action taken if the service fails to start. Possible values are:   0 - The startup (boot) program logs the error but continues the startup operation.   1 - The startup program logs the error and displays a message but continues the startup operation.   2: The startup program logs the error. If the last-known good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.   3 - The startup program logs the error, if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration. |
| TagID | Unique tag value for this service in the group that is specified by the lpLoadOrderGroup parameter. A value of zero indicates that the service has not been assigned a tag. |
| ProcessID | The current process ID for the service. This is 0 if the service is not currently running. |
| Name | The short service name, for example: MSSQLSERVER |
| BinaryPathName | The path to the executable file for the service, including startup parameters. |
| LoadOrderGroup | Pointer to a null-terminated string that names the load ordering group to which this service belongs. If the member is NULL or an empty string, the service does not belong to a load ordering group. |
| Dependencies | The short service name of services that are dependencies before this service can start. |
| ServiceStartName | The logon account for the service or LocalSystem. |
| DisplayName | The full display name for the service, for example: SQL Server (MSSQLSERVER). |
| Description | The full description for the service. |

Examples

The following example code shows a task that uses the probe to generate data about the default SQL service:

<Task ID="Microsoft.Samples.GetSQLService" Accessibility="Internal" Target="Microsoft.Samples.ApplicationX">

<Category>Operations</Category>

<ProbeAction ID="ServiceProbe" TypeID="Windows!Microsoft.Windows.ExistingWin32ServiceInformationProbe">

<ComputerName>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</ComputerName>

<ServiceName>MSSQLSERVER</ServiceName>

</ProbeAction>

</Task>

The preceding example causes the probe to output data about the default instance of SQL Server. Data is output only if the service does not exist.

Related Modules

There are a number of other module types that use this module type as part of a composition that might be more suitable to your scenario. You should use these specialized module types where it is possible rather than declaring your own compositions:

|  |  |  |
| --- | --- | --- |
| Module | Defined In | Description |
| Microsoft.Windows.ExistingWin32ServiceInformationProbe | Windows.Library | The same functionality as the Microsoft.Windows.Win32ServiceInformationProbe except that no data item is output when the service is not installed. |
| Microsoft.Windows.ServiceControlManager.QueryService | Windows.Library | Exactly the same functionality as the Microsoft.Windows.Win32ServiceInformationProbe except both parameters are defined as overrideable. |
| Microsoft.Windows.Win32ServiceDiscoveryProvider | Windows.Library | Timed discovery provider. This runs on a timed interval that is specified as configuration and ensures no cookdown happens to the workflow. The data output is the same as Microsoft.Windows.ExistingWin32ServiceInformationProbe. |
| Microsoft.Windows.ExistingWin32ServiceDiscoveryProvider | Windows.Library | The same functionality as the Microsoft.Windows.Win32ServiceDiscoveryProvider module type except that no data item is output when the service is not installed. |
| Microsoft.Windows.Win32ServiceInformationProviderWithClassSnapshotDataMapper | Windows.Library | Outputs discovery data by adding a discovery data mapper to the workflow provided by Microsoft.Windows.Win32ServiceDiscoveryProvider. The instance mapping configuration must be provided as configuration. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.RegistryProbe

You can use the Microsoft.Windows.RegistryProbe probe module type when you want to gather data from the Windows registry on a monitored computer. The module gathers one or more keys or values from the registry and provides them as a single Microsoft.Windows.RegistryData data item. This module supports retrieving any key or value from the HKEY\_LOCAL\_MACHINE hive of the registry only.

Parameters

This module accepts the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ComputerName | String | The name of the computer from which to query the registry |
| RegistryAttributeDefinitions | RegistryAttributeDefinitionsType | The set of keys and values to gather from the registry |

ComputerName

The ComputerName configuration is most commonly passed to the module by using the $Target notation, although it can be specified as any string. The module attempts to connect to either the local computer’s registry or a remote computer’s registry, depending on the configuration that is passed. For a workflow that is targeted at Windows Computer, the ComputerName is specified in the following code example:

<ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

If the target is a class that is hosted by a computer, hosting references are added as required. For example:

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

RegistryAttributeDefinitions

The RegistryAttributeDefinitions element specifies a collection of registry keys and values to collect. For each key or value a RegistryAttributeDefinitions element is specified. This configuration element has four required subelements:

|  |  |  |
| --- | --- | --- |
| Element | Type | Description |
| AttributeName | ID | An identifier for the value in the output data item. This must be unique within the RegistryAttributeDefinitions element and must consist only of letters, numbers, the underscore, hyphen, and period. Digits, the hyphen and the period cannot be used to start the name. |
| Path | String | The path to collect in the registry under HKEY\_LOCAL\_MACHINE– this path can be a key or a value. |
| PathType | Integer | Specifies whether the path is a key or a value.   0 - Specifies that the path is to a key, for which the existence of the key should be checked.   1 - Specifies that the path is to a value that should be retrieved. |
| AttributeType | Integer | The data type of the value to return:   0 - Boolean   1 - String   2 - Integer   3 - Float  Values 1, 2 and 3 are used only when the PathType is set to 1. |

The following code example of RegistryAttributeDefinitions returns the existence of a registry key:

<RegistryAttributeDefinition>

<AttributeName>ApplicationXExists</AttributeName>

<Path>SOFTWARE\OpsMgrSamples\ApplicationX</Path>

<PathType>0</PathType>

<AttributeType>0</AttributeType>

</RegistryAttributeDefinition>

The following code example of RegistryAttributeDefinitions returns a registry value: :

<RegistryAttributeDefinition>

<AttributeName>ApplicationXVersion</AttributeName>

<Path>SOFTWARE\OpsMgrSamples\ApplicationX\Version</Path>

<PathType>1</PathType>

<AttributeType>1</AttributeType>

</RegistryAttributeDefinition>

Remarks

The Microsoft.Windows.RegistryProbe module is most frequently used in discovery workflows, but it can be used for monitoring purposes in addition to tasks.

The module can collect only values from the HKEY\_LOCAL\_MACHINE hive of the Windows registry.

For multi-string values, only the first string is returned by the probe.

Examples

The following code example shows an example of a task that uses the probe module to return a set of values from the registry:

<Task ID="Microsoft.Samples.GetRegistryData" Accessibility="Internal" Target="Microsoft.Samples.ApplicationX">

<Category>Operations</Category>

<ProbeAction ID="RegistryProbe" TypeID="Windows!Microsoft.Windows.RegistryProbe">

<ComputerName>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</ComputerName>

<RegistryAttributeDefinitions>

<RegistryAttributeDefinition>

<AttributeName>ApplicationXExists</AttributeName>

<Path>SOFTWARE\OpsMgrSamples\ApplicationX</Path>

<PathType>0</PathType>

<AttributeType>0</AttributeType>

</RegistryAttributeDefinition>

<RegistryAttributeDefinition>

<AttributeName>ApplicationXVersion</AttributeName>

<Path>SOFTWARE\OpsMgrSamples\ApplicationX\Version</Path>

<PathType>1</PathType>

<AttributeType>1</AttributeType>

</RegistryAttributeDefinition>

<RegistryAttributeDefinition>

<AttributeName>ApplicationXPath</AttributeName>

<Path>SOFTWARE\OpsMgrSamples\ApplicationX\Path</Path>

<PathType>1</PathType>

<AttributeType>1</AttributeType>

</RegistryAttributeDefinition>

</RegistryAttributeDefinitions>

</ProbeAction>

</Task>

This results in the following data item:

<DataItem type="MOM.RegistryData" time="2008-10-08T13:08:26.9989812-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<Values>

<ApplicationXExists VariantType="11" Type="Boolean">true</ApplicationXExists>

<ApplicationXPath VariantType="8">c:\ApplicationX</ApplicationXPath>

<ApplicationXVersion VariantType="8">2.0</ApplicationXVersion>

</Values>

</DataItem>

Related Modules

Common discovery and monitoring scenarios are covered by other composite module types that use the registry probe module. The following table summarizes these module types, which are all data source module types:

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.FilteredRegistryDiscoveryProvider | Windows.Library | Collects registry data and maps to discovery data on a timed interval. This type takes an expression to filter data on. |
| Microsoft.Windows.RegistryDiscoveryProvider | Windows.Library | Collects registry data and maps to discovery data on a timed interval. This type takes an expression to filter data on. |
| Microsoft.Windows.RegistryDiscoveryProvider | Windows.Library | Collects registry data and maps to discovery data on a timed interval. No filtering of data is provided in this module type. This should be used only when you know that the registry keys and values will always be present. |
| Microsoft.Windows.Discovery.RegistryProviderSingle | Windows.Library | Discovers a single registry value only. This type has simpler configuration, but only a single value or key can be specified. |
| Microsoft.Windows.RegistryProvider | Windows.Library | Collects discovery data on a timed interval. This data is not mapped to discovery data. This module type should be used when registry data is not being used in discovery workflows. |
| Microsoft.Windows.RegistryProviderSingle | Windows.Library | Collects single registry value only. This data is not mapped to discovery data. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.ScriptProbeAction

The Microsoft.Windows.ScriptProbeAction probe action module type is used to execute a script as part of a monitoring workflow (a rule or a monitor) or a task workflow. The module type accepts any type of data, including trigger, and outputs System.CommandOutput data. This module uses Cscript.exe to execute the scripts.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ScriptName | String | Contains the file name of the script (including the extension) that identifies the script type. |
| Arguments | String | Contains the arguments to be supplied to the script on execution. |
| ScriptBody | String | Contains the script contents. If the script contains XML characters that must be escaped, it is recommended that the entire script be wrapped in a CDATA element. |
| SecureInput | String[256] | Contains the data to be passed to the script in a secure method. This is used to provide passwords to the script to be used during execution. |
| TimeoutSeconds | Integer | Specifies the time the script is allowed to execute before being closed by the module and marked as failed. |

The Arguments and TimeoutSeconds parameters are marked as overrideable.

ScriptName

The ScriptName parameter specifies the file name of the script to be executed. The module caches a local copy of the script on the agent that is executing the script in the health service cache.

The filename should be a legal Windows file name, and must contain the correct extension for the script type; for example.vbs for VBScript and.js for JScript. It is a best practice to name the script something meaningful so that it can be located in the health service cache on an agent computer if you need to debug execution.

Note

The script name should not contain spaces.

Arguments

The Arguments element specifies the command-line arguments that are passed to the script when it is executed. This configuration can consist of static values or dynamic references by using $Target or $Data references.

The script has no context of the object being executed against or the data that triggered the execution. Passing data into the script is the only method to provide this data. Arguments can be queried by using the WScript object as required.

If there can be spaces in your values that are passed to the script, you should be sure to enclose the values in quotes when you specify them to avoid their being split into multiple arguments by the Windows Script Host.

The following simple example code passes the computer name and the version of the application that the workflow is targeted at:

<Arguments>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer"]/NetworkName$ $Target/Property[Type='Microsoft.Samples.ApplicationX']/Version$</Arguments>

The following code is an example of the command line that is actually called when the script is executed:

CScript.exe MyScript.vbs computer1.mydomain.local 2.0

If there is a possibility that there will be spaces in your argument values, you should wrap the arguments in quotes so the script interprets them as a single argument.

In the following example, the InstallPath property is wrapped in quotes:

<Arguments>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer"]/NetworkName$ ‘$Target/Property[Type='Microsoft.Samples.ApplicationX']/Install Path$’</Arguments>

ScriptBody

The ScriptBody parameter contains the actual script text. It is a best practice to wrap the script in a CDATA block, as shown in the preceding example, to avoid having to escape certain characters that are invalid in the XML.

SecureInput

The SecureInput parameter is primarily used to pass secure data, such as passwords, into the script. Passing this data in the Arguments parameter is not secure and should not be done.

Passwords are passed by using $RunAs syntax.

TimeoutSeconds

The TimeoutSeconds parameter specifies the maximum length of time that the script can execute before it is terminated. If the script exceeds this time out and is terminated, the module reports a fatal error on execution. For scripts that are used as part of rule or monitor workflows, the time-out value should be less than the frequency with which the script executes on a given instance.

EventPolicy

The EventPolicy parameter defines the expected output and the behavior based on unexpected output from the script. For the Microsoft.Windows.ScriptProbeAction module type, a System.CommandOutput data type is the expected output. This data type contains the standard output (StdOut), the standard error (StdErr), and the exit code of the script that executed.

The following code shows an example System.CommandOutput data item:

<DataItem type="System.CommandOutput" time="2008-10-10T12:48:57.0477521-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<StdOut>This is some output text</StdOut>

<StdErr>This is some error text</StdErr>

<ExitCode>0</ExitCode>

<ProcessError></ProcessError>

</DataItem>

The event policy determines the behavior of the module, depending on the output that is provided by the script, and it is only required to be configured if you want to override the default behavior.

The script module can return one of the following four states on execution:

 Success

 Warning

 Data Loss

 Fatal Error

In the case of a task that uses this module, the default behavior is that all states except fatal error are marked as being successful task execution. This means that a script could fail during execution of a task, but Operations Manager would report the task as succeeding, which can be confusing to the user. Therefore, there are situations where it can be required to make the module return a fatal error even though the script has executed and has been completed. The EventPolicy parameter controls this behavior. The EventPolicy parameter has four elements:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Severity | Integer (0 to 2) | The state of the module to return, based on matching StdOut, StdErr, and exit codes.  The possible severity values and their meaning are:   0 - Warning   1 – Data Loss   2 – Fatal Error |
| StdOutMatches | PolicyExpression | Regular expression to match against data returned to standard out. |
| StdErrMatches | PolicyExpression | Regular expression to match against data returned to standard error. |
| ExitCodeMatches | PolicyExpression | Regular expression to match against the exit code from the script. |

All script-based module types have a default event policy set that is applied if no specific event policy is passed as configuration. The Microsoft.Windows.ScriptProbeAction type has the following default policy:

<DefaultEventPolicy>

<StdOutMatches/>

<StdErrMatches>\a+</StdErrMatches>

<ExitCodeMatches>[^0]+</ExitCodeMatches>

</DefaultEventPolicy>

This policy specifies that the default behavior of the module type is that any data in standard error or any non-zero exit code results in the module reporting execution as data loss (a severity of 1 is the default setting). Standard output is ignored.

When you specify an event policy setting, you are overriding the default policy. This is done per field. For example, if you override the severity only, then all other defaults take effect.

To force the module to report a failure when standard out starts with the word, “Error”, set the event policy as follows:

EventPolicy>

<Severity>2</Severity>

<StdErrMatches>^Error</StdErrMatches>

</EventPolicy>

This policy forces a module failure and reports the full output of that failure as task output.

To always return task success and report standard output, standard error, and exit code as task output, set the policy as follows:

<EventPolicy>

<Severity>0</Severity>

<StdErrMatches>\a+</StdErrMatches>

</EventPolicy>

In the case of a rule or monitor workflow where the Microsoft.Windows.ScriptProbeAction module is used, the following behavior occurs for each non-success state:

 Warning – the workflow continues executing and uses the data item as output from the script module. Event 21413 is logged from the Health Service Modules source in the Operations Manager event log, giving full context to the warning.

 Data Loss – the data item is dropped from the workflow and no subsequent modules execute. This means that the monitor state is not set in the case of a monitor. The workflow continues to execute and is not unloaded. Event 21413 is logged from the Health Service Modules source in the Operations Manager event log, giving full context to the failure.

 Fatal Error – the module is failed, and the workflow that had the failure is unloaded and does not run again until new configuration is deployed or the agent is restarted. Event 21413 is logged, followed by Event 1103 from the HealthService source, indicating that the workflow was unloaded.

Note

The default settings for the standard error and the standard output match are case insensitive and match regular expression.

There are two optional attributes that can be set on the StdOutMatches, StdErrMatches and ExitCodeMatches elements:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| Operator | PolicyOperatorType | This can be set to MatchesRegularExpression and DoesNotMatchRegularExpression. Matches is the default value if this parameter is not specified. |
| CaseSensitve | Boolean | Indicates whether the match should be case-sensitive. A value of true indicates a case sensitive match only. A value of false is the default if this parameter specified. |

An example event policy configuration that uses an optional attribute is shown below:

<DefaultEventPolicy>

<StdOutMatches Operator="DoesNotMatchRegularExpression"><![CDATA[<DataItem.+/DataItem\b\*>}|{<DataItem.\*/>}]]></StdOutMatches>

<StdErrMatches>\a+</StdErrMatches>

<ExitCodeMatches>[^0]+</ExitCodeMatches>

</DefaultEventPolicy>

Remarks

A probe module type should not be used when any aspect of the system is being changed; instead, there is a corresponding write action for this purpose. The script that is executed can be any script type that is supported by the Windows Script Host (WSH) on the target computers. The type of script is identified by the file name extension that is provided in the configuration. The two common types of script are VBScript (.vbs extension) and JScript (.js extension).

For VB Script, you can write to standard output in two ways:

WScript.Echo "This is some output"

…or…

WScript.Echo "This is some output"

To write to standard error:

WScript.StdErr.Write "This is some error text"

To return a non-zero exit code, you must call the Quit function with the required error code. For example:

WScript.Quit(2)

For more detailed information about writing to standard out and standard error, see the [Windows Script Host reference documentation](http://go.microsoft.com/fwlink/?LinkId=137114).

Examples

The following code example shows the Microsoft.Windows.ScriptProbeAction module used in a simple task:

</Task>

<Task ID="Microsoft.Samples.TaskTest1" Accessibility="Internal" Target="Microsoft.Samples.ApplicationX">

<Category>Operations</Category>

<ProbeAction ID="Script" TypeID="Windows!Microsoft.Windows.ScriptProbeAction">

<ScriptName>Microsoft.Samples.TestScript1.vbs</ScriptName>

<Arguments>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</Arguments>

<ScriptBody>

<![CDATA[

Option Explicit

Dim Args

Set Args = WScript.Arguments

WScript.Echo "This is some output text"

WScript.Echo "This is a parameter passed in: " & Args(0)

WScript.StdOut.Write "This is some more output text"

WScript.StdErr.Write "This is some error text"

WScript.Quit(3)

]]>

</ScriptBody>

<SecureInput/>

<TimeoutSeconds>10</TimeoutSeconds>

<EventPolicy>

<Severity>0</Severity>

<StdErrMatches>\a+</StdErrMatches>

</EventPolicy>

</ProbeAction>

</Task>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.ScriptWriteAction | Windows.Library | Used as a task or recovery when the script is changing something on the system on which it is run. |
| Microsoft.Windows.TimedScript.DiscoveryProvider | Windows.Library | Used for scheduled discovery scripts that are triggered by the simple scheduler. |
| Microsoft.Windows.TimedScript.PropertyBagProvider | Windows.Library | Used for scheduled scripts that are used in rule or monitor workflows. This module type provides property bag output that is better suited for XPath referencing and expression evaluation. This module is triggered by the simple scheduler. |
| Microsoft.Windows.TimedScript.EventProvider | Windows.Library | Used for scheduled scripts that output data that is mapped to an event and stored in the Operations Manager databases. |
| Microsoft.Windows.TimedScript.PerformanceProvider | Windows.Library | Used for scheduled scripts that output data that is to be mapped to performance data and stored in the Operations Manager databases. |
| Microsoft.Windows.ScriptGenerated.EventProvider | Windows.Library | Used as a data source to collect events that are generated by the MOM.ScriptAPI in other scripts. |
| Microsoft.Windows.ScriptDiscoveryProbe | Windows.Library | Used for discovery scripts that are triggered by data sources other than the simple scheduler. |
| Microsoft.Windows.ScriptPropertyBagProbe | Windows.Library | Used for monitoring scripts that are triggered by data sources other than the simple scheduler. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.Win32ServiceInformationProbe

The Microsoft.Windows.Win32ServiceInformationProbe probe action module type is used to query Windows service information about a specific service from the Service Control Manager. A module of this type takes a trigger as input, which means it can accept any data type to trigger execution. The module returns a System.PropertyBagData type that contains all the properties of the service. This data can be used in monitoring workflows or discovery workflows.

Parameters

The Microsoft.Windows.Win32ServiceInformationProbe probe action module type accepts a System.PropertyBag data type with the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ComputerName | String | The name of the computer to query. |
| ServiceName | String | The service name to query. |

The ComputerName configuration is most commonly passed to the module by using the $Target notation, although it can be specified as any string.

The module attempts to connect to either the local computer’s Service Control Manager or to a remote computer’s Service Control Manager, depending on the configuration that is passed.

If the target is a class that is hosted by computer hosting, references are added as required. For example:

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

The service name specifies the short name of the Windows service to collect data on. This can be found from the properties page of the Services MMC snap-in. For example:

<ServiceName>spooler</ServiceName>

Remarks

The Microsoft.Windows.Win32ServiceInformationProbe module outputs data even if the Windows Service that is being queried does not exist. This allows the management pack author to check whether a service exists in monitoring scenarios. This behavior might not be wanted, particularly in the case of discovery. You should use the Microsoft.Windows.ExistingWin32ServiceInformationProbe instead if you do not want data output when the service does not exist.

The module outputs a property bag object that contains data about the service that was queried. The following code example shows an example data item:

<DataItem type="System.PropertyBagData" time="2008-10-08T15:52:27.0027578-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<Property Name="State" VariantType="3">4</Property>

<Property Name="ServiceType" VariantType="3">16</Property>

<Property Name="StartMode" VariantType="3">2</Property>

<Property Name="ErrorControl" VariantType="3">1</Property>

<Property Name="TagId" VariantType="3">0</Property>

<Property Name="ProcessId" VariantType="3">1880</Property>

<Property Name="Name" VariantType="8">MSSQLSERVER</Property>

<Property Name="BinaryPathName" VariantType="8">"C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\sqlservr.exe" -sMSSQLSERVER</Property>

<Property Name="LoadOrderGroup" VariantType="8"></Property>

<Property Name="Dependencies" VariantType="8"></Property>

<Property Name="ServiceStartName" VariantType="8">LocalSystem</Property>

<Property Name="DisplayName" VariantType="8">SQL Server (MSSQLSERVER)</Property>

<Property Name="Description" VariantType="8">Provides storage, processing and controlled access of data and rapid transaction processing.</Property>

</DataItem>

The data item is a property bag that contains the following properties:

|  |  |
| --- | --- |
| Property | Description |
| State | The current service status. Possible values are:   0 - Unknown   1 - Stopped   2 – Start pending   3 – Stop pending   4 - Running   5 – Continue pending   6 – Pause pending   7 - Paused   8 – Service not found   9 – Server not found |
| ServiceType | The type of service. Possible values are:   1 – Kernel/driver   2 – File system driver   16 – Own process   32 – Share process |
| StartMode | The startup type of the service. Possible values are:   0 – Boot start   1 – System start   2 - Automatic   3 - Manual   4 - Disabled |
| ErrorControl | Severity of the error and the action taken if the service fails to start. Possible values are:   0 - The startup (boot) program logs the error but continues the startup operation.   1 - The startup program logs the error and displays a message but continues the startup operation.   2: The startup program logs the error. If the last-known good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known good configuration.   3 - The startup program logs the error, if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration. |
| TagID | Unique tag value for this service in the group that is specified by the lpLoadOrderGroup parameter. A value of zero indicates that the service has not been assigned a tag. |
| ProcessID | The current process ID for the service. This is 0 if the service is not currently running. |
| Name | The short service name, for example: MSSQLSERVER. |
| BinaryPathName | The path to the executable file for the service, including startup parameters. |
| LoadOrderGroup | Pointer to a null-terminated string that names the load ordering group to which this service belongs. If the member is NULL or an empty string, the service does not belong to a load ordering group. |
| Dependencies | The short service name of services that are dependencies before this service can start. |
| ServiceStartName | The logon account for the service or LocalSystem. |
| DisplayName | The full display name for the service, for example: SQL Server (MSSQLSERVER). |
| Description | The full description for the service. |

The following example shows an example data item when the service is not installed:

<DataItem type="System.PropertyBagData" time="2008-11-10T08:31:07.2117298-08:00" sourceHealthServiceId="FAA43A3F-E644-3DC4-D637-406D3E94A26A">

<Property Name="State" VariantType="3">8</Property>

<Property Name="ServiceType" VariantType="3">0</Property>

<Property Name="StartMode" VariantType="3">0</Property>

<Property Name="ErrorControl" VariantType="3">0</Property>

<Property Name="TagId" VariantType="3">0</Property>

<Property Name="ProcessId" VariantType="3">0</Property>

<Property Name="Name" VariantType="8">MSSQLSERVER</Property>

<Property Name="BinaryPathName" VariantType="8"/>

<Property Name="LoadOrderGroup" VariantType="8"/>

<Property Name="Dependencies" VariantType="8"/>

<Property Name="ServiceStartName" VariantType="8"/>

<Property Name="DisplayName" VariantType="8"/>

<Property Name="Description" VariantType="8"/>

</DataItem>

Examples

The following example code shows a task that uses the probe to generate data about the default SQL service:

<Task ID="Microsoft.Samples.GetSQLService" Accessibility="Internal" Target="Microsoft.Samples.ApplicationX">

<Category>Operations</Category>

<ProbeAction ID="ServiceProbe" TypeID="Windows!Microsoft.Windows.Win32ServiceInformationProbe">

<ComputerName>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</ComputerName>

<ServiceName>MSSQLSERVER</ServiceName>

</ProbeAction>

</Task>

The preceding example causes the probe to output data about the default instance of SQL Server.

Related Modules

There are a number of other module types that use this module type as part of a composition that might be more suitable to your scenario. You should use these specialized module types where it is possible rather than declaring your own compositions:

|  |  |  |
| --- | --- | --- |
| Module | Defined In | Description |
| Microsoft.Windows.ExistingWin32ServiceInformationProbe | Windows.Library | The same functionality as the Microsoft.Windows.Win32ServiceInformationProbe except that no data item is output when the service is not installed. |
| Microsoft.Windows.ServiceControlManager.QueryService | Windows.Library | Exactly the same functionality as the Microsoft.Windows.Win32ServiceInformationProbe except both parameters are defined as overrideable. |
| Microsoft.Windows.Win32ServiceDiscoveryProvider | Windows.Library | Timed discovery provider. This runs on a timed interval that is specified as configuration and ensures no cookdown happens to the workflow. The data output is the same as Microsoft.Windows.ExistingWin32ServiceInformationProbe. |
| Microsoft.Windows.ExistingWin32ServiceDiscoveryProvider | Windows.Library | The same functionality as the Microsoft.Windows.Win32ServiceDiscoveryProvider module type except that no data item is output when the service is not installed. |
| Microsoft.Windows.Win32ServiceInformationProviderWithClassSnapshotDataMapper | Windows.Library | Outputs discovery data by adding a discovery data mapper to the workflow provided by Microsoft.Windows.Win32ServiceDiscoveryProvider. The instance mapping configuration must be provided as configuration. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.WmiProbe

The Microsoft.Windows.WmiProbe module is a probe action module type that is used to query Windows Management Instrumentation (WMI) for event or instance data and return the information as a System.PropertyBag type.

Parameters

The Microsoft.Windows.WmiProbe module accepts the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| NameSpace | String | The WMI namespace to query. |
| Query | String | The WMI query string. |

The NameSpace parameter must be a valid WMI namespace that is present on the monitored computer, or the module raises an error in the Operations Manager event log.

The namespace should always specify the computer to connect to as part of the string. If a computer is not specified, the workflow reads from the local computer, which does not work for agentless monitoring scenarios. The computer name is most commonly passed to the module that is using the $Target notation.

The following code example shows a fully configured NameSpace parameter in a workflow that is targeted at the Windows Computer class. This example retrieves data from the root\cimv2 WMI namespace.

<NameSpace>\\$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$\ROOT\CIMV2</NameSpace>

For a workflow that is targeted at a class hosted by a Windows computer, the following code is used:

<NameSpace>\\$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$\ROOT\CIMV2</NameSpace>

The Query parameter must be a valid WMI query that can be executed against the specified WMI namespace. For example:

<Query>select \* from win32\_logicaldisk</Query>

It is possible to reference object properties when you specify the query. The following code is a more complex example that uses the DeviceID property a class:

<Query>associators of {win32\_diskdrive='$Target/Property[Type="Windows!Microsoft.Windows.LogicalDevice"]/DeviceID$'} where ResultClass=Win32\_DiskPartition</Query>

Remarks

The WMI probe outputs one or more property-bag data items. The exact properties that are available in the output are dependent on the namespace and query that were specified.

The following code shows an example data item for a C: drive instance of the Win32\_LogicalDisk class. This was generated by using the query select \* from Win32\_LogicalDisk where DeviceID=’C:’:

<DataItem type="System.PropertyBagData" time="2008-10-26T14:34:39.0657744-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<Property Name="\_\_CLASS" VariantType="8">Win32\_LogicalDisk</Property>

<Property Name="\_\_DERIVATION" VariantType="8">CIM\_LogicalDisk,CIM\_StorageExtent,CIM\_LogicalDevice,CIM\_LogicalElement,CIM\_ManagedSystemElement</Property>

<Property Name="\_\_DYNASTY" VariantType="8">CIM\_ManagedSystemElement</Property>

<Property Name="\_\_GENUS" VariantType="3">2</Property>

<Property Name="\_\_NAMESPACE" VariantType="8">root\cimv2</Property>

<Property Name="\_\_PATH" VariantType="8">\\STWILSONX20\root\cimv2:Win32\_LogicalDisk.DeviceID="C:"</Property>

<Property Name="\_\_PROPERTY\_COUNT" VariantType="3">40</Property>

<Property Name="\_\_RELPATH" VariantType="8">Win32\_LogicalDisk.DeviceID="C:"</Property>

<Property Name="\_\_SERVER" VariantType="8">STWILSONX20</Property>

<Property Name="\_\_SUPERCLASS" VariantType="8">CIM\_LogicalDisk</Property>

<Property Name="Access" VariantType="18">0</Property>

<Property Name="Caption" VariantType="8">C:</Property>

<Property Name="Compressed" VariantType="11" Type="Boolean">false</Property>

<Property Name="CreationClassName" VariantType="8">Win32\_LogicalDisk</Property>

<Property Name="Description" VariantType="8">Local Fixed Disk</Property>

<Property Name="DeviceID" VariantType="8">C:</Property>

<Property Name="DriveType" VariantType="19">3</Property>

<Property Name="FileSystem" VariantType="8">NTFS</Property>

<Property Name="FreeSpace" VariantType="8">127203143680</Property>

<Property Name="MaximumComponentLength" VariantType="19">255</Property>

<Property Name="MediaType" VariantType="19">12</Property>

<Property Name="Name" VariantType="8">C:</Property>

<Property Name="QuotasDisabled" VariantType="11" Type="Boolean">true</Property>

<Property Name="QuotasIncomplete" VariantType="11" Type="Boolean">false</Property>

<Property Name="QuotasRebuilding" VariantType="11" Type="Boolean">false</Property>

<Property Name="Size" VariantType="8">159989886976</Property>

<Property Name="SupportsDiskQuotas" VariantType="11" Type="Boolean">true</Property>

<Property Name="SupportsFileBasedCompression" VariantType="11" Type="Boolean">true</Property>

<Property Name="SystemCreationClassName" VariantType="8">Win32\_ComputerSystem</Property>

<Property Name="SystemName" VariantType="8">STWILSONX20</Property>

<Property Name="VolumeDirty" VariantType="11" Type="Boolean">false</Property>

<Property Name="VolumeName" VariantType="8"></Property>

<Property Name="VolumeSerialNumber" VariantType="8">F4E1E93E</Property>

</DataItem>

If the query that is specified returns multiple property bag data items, the workflow must be targeting a class that has multiple instances for the data to be used, and the data must have some property in it that can associate the correct property bag with the correct Operations Manager object.

For example, if you are monitoring logical disks and a monitor is running that executes the query select \* from Win32\_LogicalDisk, the DeviceID property of the Operations Manager class would be matched to the DeviceID WMI property in an expression module to filter out the other instances of logical disks.

Examples

The following example shows the Microsoft.Windows.WmiProbe module being used in a task that uses a property of the application to get data about the disk drive that the application is installed on:

<Task ID="Microsoft.Samples.GetLogicalDiskData" Accessibility="Internal" Target="Microsoft.Samples.ApplicationX">

<Category>Operations</Category>

<ProbeAction ID="Probe" TypeID="Windows!Microsoft.Windows.WmiProbe">

<NameSpace>\\$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$\root\cimv2</NameSpace>

<Query>select \* from win32\_logicaldisk where DeviceID='$Target/Property[Type="Microsoft.Samples.ApplicationX"]/HostDrive$'</Query>

</ProbeAction>

</Task>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.WmiProvider | Microsoft.Windows.Library | Runs a scheduled WMI probe as part of a rule or monitor workflow. |
| Microsoft.Windows.WmiProviderWithClassSnapshotDataMapper | Microsoft.Windows.Library | Runs a scheduled WMI probe and maps the data to discovery data. This can be used only in a discovery workflow. |
| Microsoft.Windows.WmiProviderWithRelationshipSnapshotDataMapper | Microsoft.Windows.Library | Runs a scheduled WMI probe and maps the data to relationship discovery data. This can be used only in a discovery workflow. |
| Microsoft.Windows.WmiProvider.EventProvider | Microsoft.Windows.Library | Runs a scheduled WMI probe as part of a rule or monitor and maps the data to event format so it can be stored in the database or used with modules that are expecting event data. |
| Microsoft.Windows.WmiEventProvider | Microsoft.Windows.Library | Provides WMI event data rather than instance data. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.ApplicationLog.GenericCSVLog.FilteredEventDataProvider

The System.ApplicationLog.GenericCSVLog.FilteredEventProvider is a data source module type that is used to provide filtered events from a delimited, text-based, log file and output the events as System.Event.Data.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| LogFileDirectory | String | Contains the full path to the local directory to read the log file or files from. |
| LogFilePattern | String | Contains the file name or wildcard pattern to match for the log files. |
| LogIsUTF8 | Boolean | Indicates whether the log file is UTF8-encoded. The value should be either true or false. |
| Separator | String | Specifies the character that is used in the file as a delimiter. |
| Expression | ExpressionType | Contains the filter expression to apply to the log file entries. |

The parameters for this module type are configured in the same way as the [System.ApplicationLog.GenericLog.FilteredEventProvider](#z27a5acba882b4dffaedcb36a01cb2e61) module type. There is one additional parameter called Separator that specifies the delimiter to use. This delimiter separates the text of a single log entry into multiple fields. Commonly, a comma is used, but any single character can be used.

Remarks

The behavior of this module type is almost identical to the System.ApplicationLog.GenericLog.FilteredEventProvider module type. For more information about module behavior, see to the documentation of this module type.

The data item that is output before the expression is applied is of type System.ApplicationLog.GenericLogEntryData. Unlike the data item that is output for the non-delimited module type, multiple parameters are available in the data item that represents the delimited fields. For example, consider the following entry in the log file that is defined as follows:

Error,101,Fatal Crash,Component X

The data item that the preceding expression would evaluate over contains four parameters as shown in the following code example:

<DataItem type="System.ApplicationLog.GenericLogEntryData" time="2008-10-22T21:31:25.4252998-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<LogFileDirectory>C:\Program Files\ApplicationX</LogFileDirectory>

<LogFileType>Generic CSV Log File Format</LogFileType>

<LogFileName>C:\Program Files\ApplicationX\AppXLog.csv</LogFileName>

<Params>

<Param>Error</Param>

<Param>101</Param>

<Param>Fatal Crash</Param>

<Param>Component X</Param>

</Params>

</DataItem>

The data that is written to the database is event data. The following code example shows the format of a sample data item:

<DataItem type="System.Event.Data" time="2008-10-22T21:31:25.4252998-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<EventOriginId>{0488804B-8468-7A9A-143B-ED7CC7317E4A}</EventOriginId>

<PublisherId>{B7E5370B-0F54-9669-8EFF-D12198659BFF}</PublisherId>

<PublisherName>GenericCSVLog</PublisherName>

<EventSourceName>GenericCSVLog</EventSourceName>

<Channel>GenericCSVLog</Channel>

<LoggingComputer></LoggingComputer>

<EventNumber>0</EventNumber>

<EventCategory>3</EventCategory>

<EventLevel>0</EventLevel>

<UserName></UserName>

<RawDescription></RawDescription>

<CollectDescription Type="Boolean">true</CollectDescription>

<EventData>

<DataItem type="System.ApplicationLog.GenericLogEntryData" time="2008-10-22T21:31:25.4252998-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<LogFileDirectory>C:\Program Files\ApplicationX</LogFileDirectory>

<LogFileType>Generic CSV Log File Format</LogFileType>

<LogFileName>C:\Program Files\ApplicationX\AppXLog.csv</LogFileName>

<Params>

<Param>Error</Param>

<Param>101</Param>

<Param>Fatal Crash</Param>

<Param>Component X</Param>

</Params>

</DataItem>

</EventData>

<EventDisplayNumber>0</EventDisplayNumber>

<EventDescription></EventDescription>

</DataItem>

Examples

The following code example shows a rule that uses the System.ApplicationLog.GenericCSVLog.FilteredEventProvider data source to collect all log entries from a text log file when the first field matches the string Error and the second field matches either 101 or 102.

<Rule ID="Microsoft.Samples.LogFileCSVTest" Target="Microsoft.Samples.ApplicationX">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Log" TypeID="AppLog!System.ApplicationLog.GenericCSVLog.FilteredEventProvider">

<LogFileDirectory>%ProgramFiles%\ApplicationX</LogFileDirectory>

<LogFilePattern>AppXLog.csv</LogFilePattern>

<LogIsUTF8>false</LogIsUTF8>

<Separator>,</Separator>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">Error</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<RegExExpression>

<ValueExpression>

<XPathQuery Type="Integer">Params/Param[2]</XPathQuery>

</ValueExpression>

<Operator>MatchesRegularExpression</Operator>

<Pattern>^(101|102)$</Pattern>

</RegExExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishEventData"/>

</WriteActions>

</Rule>

The following code example shows an alert-generating rule that generates an alert for any log entry that has “Critical” in the first field. The alert is suppressed on the value in the second field. For more information about the alert generation module, see the System.GenerateAlert documentation.

<Rule ID="Microsoft.Samples.LogFileCSVAlert" Target="Microsoft.Samples.ApplicationX" Enabled="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Log" TypeID="AppLog!System.ApplicationLog.GenericCSVLog.FilteredEventProvider">

<LogFileDirectory>%ProgramFiles%\ApplicationX</LogFileDirectory>

<LogFilePattern>AppXLog.csv</LogFilePattern>

<LogIsUTF8>false</LogIsUTF8>

<Separator>,</Separator>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">Critical</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="GenerateAlert" TypeID="SystemHealth!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>1</Severity>

<AlertMessageId>$MPElement[Name="Microsoft.Samples.LogFileCSVAlert.AlertMessage"]$</AlertMessageId>

<AlertParameters>

<AlertParameter1>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</AlertParameter1>

<AlertParameter2>$Data/EventData/DataItem/Params/Param[2]$</AlertParameter2>

</AlertParameters>

<Suppression>

<SuppressionValue>$Data/EventData/DataItem/Params/Param[2]$</SuppressionValue>

</Suppression>

</WriteAction>

</WriteActions>

</Rule>

Note that in the preceding example the second parameter is referenced by the expression $Data/EventData/DataItem/Params/Param[2]$ in the alert module. The log file entry data item has been mapped to an event by the time it is passed to the alert module, so the XPathQuery must use the EventData that is generated by the alert to get to the second parameter.

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Define In | Usage |
| System.ApplicationLog.GenericCSVLog.EventProvider | System.ApplicationLog.Library | Provides unfiltered event data from delimited log file entries. |
| System.ApplicationLog.GenericCSVLogReader | System.ApplicationLog.Library | Provides unfiltered data from delimited log file entries without mapping to event data. The output cannot be stored directly in the database, but it can be used as input to further modules. |
| System.ApplicationLog.GenericLog.FilteredEventProvider | System.ApplicationLog.Library | Use as an alternative to the filtered comma-separated value (CSV) event provider when the log file does not have delimited fields. |
| System.ApplicationLog.GenericLog.EventProvider | System.ApplicationLog.Library | Use as an alternative to the unfiltered CSV event provider when the log file does not have delimited fields. |
| System.ApplicationLog.GenericLogReader | System.ApplicationLog.Library | Use as an alternative to the unfiltered CSV provider when the log file does not have delimited fields. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.ApplicationLog.GenericLog.FilteredEventProvider

The System.ApplicationLog.GenericCSVLog.FilteredEventProvider is a data source module type that is used to provide filtered events from a delimited, text-based log file and output the events as System.Event.Data.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| LogFileDirectory | String | Contains the full path to the local directory to read the log file or files from. |
| LogFilePattern | String | Contains the file name or wildcard pattern to match for the log files. |
| LogIsUTF8 | Boolean | Indicates whether the log file is UTF8-encoded. The value should be either true or false. |
| Expression | ExpressionType | Contains the filter expression to apply to the log file entries. |

LogFileDirectory

The LogFileDirectory parameter specifies an absolute path to the log file or files or to a path by using environment variables or $Target variables or both. Wildcards cannot be used in this parameter, because a single workflow can look for files only in a single directory by using this module type. Absolute paths are generally not recommended, because many applications can be installed to non-default directories by the user.

The following code shows an example of an absolute file path:

<LogFileDirectory>C:\ApplicationX</LogFileDirectory>

The following code shows the use of an environment variable:

<LogFileDirectory>%ProgramFiles%\ApplicationX</LogFileDirectory>

The following code shows the use of a discovered property of an object:

<LogFileDirectory>$Target/Property[Type='Microsoft.Samples.ApplicationX']/Path$</LogFileDirectory>

LogFilePattern

The LogFilePattern parameter can contain an absolute file name reference. For example:

<LogFilePattern>AppX.log</LogFilePattern>

In the preceding example, only a single log file is ever used as the source for the module. The module maintains a bookmark to the current position of the log file and acts only on new log file entries.

The parameter can also contain a wildcard reference to file names. For example:

<LogFilePattern>AppX\*.log</LogFilePattern>

This parameter is designed to work with applications that create new log files when the current log file gets to a certain size, or when a new file is created based on the advancement of time, such as a new file that is created every day.

It is important to note that the module maintains a bookmark only to the most recent file and is designed to work with only a single file at a time. If there are multiple files being written to in alternating order, the module does not output data as expected.

Expression

The expression that is specified in this parameter is used to filter the log file data that is output from the log reader module. The data type that is passed to the expression module built into the System.ApplicationLog.GenericLog.FilteredEventProvider module type is of type System.ApplicationLog.GenericLogEntryData.

The following code shows an example data item:

<DataItem type="System.ApplicationLog.GenericLogEntryData" time="2008-10-22T18:05:50.7802742-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<LogFileDirectory>C:\Program Files\ApplicationX</LogFileDirectory>

<LogFileType>Generic Log File Format</LogFileType>

<LogFileName>C:\Program Files\ApplicationX\test1.log</LogFileName>

<Params>

<Param><![CDATA[This is a full line of text from the log file]]></Param>

</Params>

</DataItem>

Each new line of text in the text file is output as a single data item. The log file name and directory are contained in the data item and can be used in the expression module, together with the actual log file entry. The following code shows an example expression that looks for a substring in the contents of the log file entry:

<Expression>

<RegExExpression>

<ValueExpression>

<XPathQuery Type="String">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>ContainsSubstring</Operator>

<Pattern>Warning:</Pattern>

</RegExExpression>

</Expression>

In the preceding example only entries that contain “Warning:” in their entry are output by the module. The Expression parameter is fully described in the [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) module type documentation.

Remarks

This module type expects log files that contain a separate line for each new log file entry. If each line has a delimiter to separate fields (such as a comma), there are alternative modules that can be used. For more information, see the Related Modules section later in this document.

This module type outputs log file entries in the form of event data so that they can be written to the Operations Manager database and data warehouse.

The following code shows an example data item that is written to the database:

<DataItem type="System.Event.Data" time="2008-10-22T18:36:51.3559553-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<EventOriginId>{0488804B-8468-7A9A-143B-ED7CC7317E4A}</EventOriginId>

<PublisherId>{B7E5370B-0F54-9669-8EFF-D12198659BFF}</PublisherId>

<PublisherName>GenericLog</PublisherName>

<EventSourceName>GenericLog</EventSourceName>

<Channel>GenericLog</Channel>

<LoggingComputer></LoggingComputer>

<EventNumber>0</EventNumber>

<EventCategory>3</EventCategory>

<EventLevel>0</EventLevel>

<UserName></UserName>

<RawDescription></RawDescription>

<CollectDescription Type="Boolean">true</CollectDescription>

<EventData>

<DataItem type="System.ApplicationLog.GenericLogEntryData" time="2008-10-22T18:36:51.3559553-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<LogFileDirectory>C:\Program Files\ApplicationX</LogFileDirectory>

<LogFileType>Generic Log File Format</LogFileType>

<LogFileName>C:\Program Files\ApplicationX\test2.log</LogFileName>

<Params>

<Param><![CDATA[Warning:Some warning has occurred! ]]></Param>

</Params>

</DataItem>

</EventData>

<EventDisplayNumber>0</EventDisplayNumber>

<EventDescription></EventDescription>

</DataItem>

On the first initiation of the workflow, the module looks for the most recent file that matches the file pattern in the specified directory and outputs the next entry that matches the expression criteria. It does not read the contents of the existing file.

When a newer file that matches the file pattern is created, the module switches to gather events from the newer file. At any time, the module holds a bookmark to only a single file.

The module does not support log-file wrapping.

Examples

The following example shows a rule that looks for a substring of “Warning:” or “Error:” in a log file and writes the data as an event to the Operations Manager database and the data warehouse.

<Rule ID="Microsoft.Samples.LogFileTest" Target="Microsoft.Samples.ApplicationX" Enabled="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Log" TypeID="AppLog!System.ApplicationLog.GenericLog.FilteredEventProvider">

<LogFileDirectory>%ProgramFiles%\ApplicationX</LogFileDirectory>

<LogFilePattern>ApplicationX.log</LogFilePattern>

<LogIsUTF8>false</LogIsUTF8>

<Expression>

<Or>

<Expression>

<RegExExpression>

<ValueExpression>

<XPathQuery Type="String">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>ContainsSubstring</Operator>

<Pattern>Warning:</Pattern>

</RegExExpression>

</Expression>

<Expression>

<RegExExpression>

<ValueExpression>

<XPathQuery Type="String">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>ContainsSubstring</Operator>

<Pattern>Error:</Pattern>

</RegExExpression>

</Expression>

</Or>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishEventData"/>

</WriteActions>

</Rule>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Define In | Usage |
| System.ApplicationLog.GenericCSVLog.EventProvider | System.ApplicationLog.Library | Provides unfiltered event data from delimited log file entries. |
| System.ApplicationLog.GenericCSVLogReader | System.ApplicationLog.Library | Provides unfiltered data from delimited log file entries without mapping to event data. The output cannot be stored directly in the database, it but can be used as input to further modules. |
| System.ApplicationLog.GenericLog.FilteredEventProvider | System.ApplicationLog.Library | Use as an alternative to the filtered comma-separated value file (CSV) event provider when the log file does not have delimited fields. |
| System.ApplicationLog.GenericLog.EventProvider | System.ApplicationLog.Library | Use as an alternative to the unfiltered CSV event provider when the log file does not have delimited fields. |
| System.ApplicationLog.GenericLogReader | System.ApplicationLog.Library | Use as an alternative to the unfiltered CSV provider when the log file does not have delimited fields. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.CommandExecuterProbe

The System.CommandExecuterProbe module is a probe action module type that is used to execute a batch file or an executable file from the command prompt. A module of this type accepts any type of data as the input data type, and it outputs System.CommandOutput data.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ApplicationName | String | Contains the full path to the executable file or the batch file to run. |
| WorkingDirectory | String | Specifies the working directory to use during execution. This can be left empty if the current directory is to be used. |
| CommandLine | String | Contains the full command-line arguments to pass to the application. |
| SecureInput | String[256] | Optional. Contains data that is to be passed to the command in a secure method. This parameter is used to provide passwords to a batch file during execution. |
| TimeoutSeconds | Integer | Specifies the time the script is allowed to execute before being closed by the module and marked as failed. |
| RequireOutput | Boolean | Defines whether output is expected from the application or batch file. If it is set to true, the module logs an error to the event log if no output is returned by the application or batch file. |
| Files | CommandExecuterFilesType | Optional. Contains one or more text-based scripts or batch files. |
| DefaultEventPolicy | CommandExecuterEventPolicyType | Contains the default event policy to be applied if no event policy is specified. This parameter is optional, and it should be set only when defining a new composite from the module type. |
| EventPolicy | CommandExecuterEventPolicyType | Optional. Controls the success or failure result of the module |

The TimeoutSeconds parameter is the only parameter that is marked as overrideable for this module type.

ApplicationName

This parameter defines the full path to the executable file that will be run. It is possible to use variable path names (such as %windir%) as part of the application path. The parameter can contain static values or use $Target values, as required.

The following code shows an example that executes IPConfig:

<ApplicationName>%WINDIR%\System32\IPCONFIG.EXE</ApplicationName>

WorkingDirectory

The WorkingDirectory parameter specifies the directory that is used when starting the application. If the full path is not specified in the ApplicationName parameter, this directory must be specified.

This parameter is required as part of the configuration, but it can be empty.

CommandLine

The value that is specified in CommandLine is passed to the application that is specified in the ApplicationName parameter as command-line parameters. This element can contain static values or $Target values.

SecureInput

The SecureInput parameter is primarily used to pass secure data, such as passwords, into the application or batch file. Passing this data in the CommandLine parameter is not secure and should not be done. Passwords are passed by using $RunAs syntax.

TimeoutSeconds

The TimeoutSeconds parameter specifies the maximum length of time that the command can run before being halted. If the command exceeds this timeout and is halted, the module reports a data loss on execution and an event is logged to the local Operations Manager event log.

For commands that are run as part of rule or monitor workflows, the timeout value should be less that the frequency with which the command runs on for a given instance.

RequireOutput

The RequireOutput parameter specifies whether output is required from the command. In monitoring and discovery workflows, this is parameter is almost always set to true. The workflow expects output and raises an error event in the local Operations Manager event log if no output is generated, depending on the event policy that is defined.

Files

The optional Files parameter contains a list of text-based files, which could be scripts, batch files, Power Shell scripts or any type of file content. These files are put in the directory from which the command is run. The directory is under the Health Service State directory that is used by the health service. All files that form a single module instance are extracted into a single directory.

To start the command in the directory where the files are extracted, leave the working directory empty. The files are available to be used as required in your probe module.

One or more files can be specified and each must have three parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Name | String | Contains the file name of the file that is created. |
| Contents | String | Contains the contents of the file. |
| Unicode | Boolean | Optional. Specifies whether the contents of the file are Unicode. This parameter defaults to true. Because the command interpreter understands only ASCII files, this parameter should be set to false if you are extracting a .cmd or .bat file. |

The following code shows a simple example of the Files parameter:

<Files>

<File>

<Name>StartService.cmd</Name>

<Contents>

<![CDATA[

if "%OSVer%"=="W2K" goto Win2K

sc \\%1 start %2

goto end

:Win2K

net start %2

goto end

:end

]]>

</Contents>

<Unicode>false</Unicode>

</File>

</Files>

Files can be referenced in the command-line arguments by using $File/Name$ syntax. In the preceding example, this file would be referenced as $File/StartService.cmd$.

The file is put into the directory from which the application is run f unless a working directory is specified. It is not necessary to use $File; the file can be referenced with only its name.

For example, the command line for the preceding example can be defined as:

<CommandLine>StartService.cmd $Target/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$ "$Config/ServiceName$"</CommandLine>

DefaultEventPolicy

The DefaultEventPolicy parameter contains the event policy to apply to the output of the module if no EventPolicy is defined. This is only required when you are building a composite module type.

If the System.CommandExecuterProbe is used directly in a workflow, then the EventPolicy parameter should be set instead.

The example below defines a policy that causes the workflow to throw an error when the output does not contain a data item. The standard error contains any data, or the standard error contains the exit code if it is not zero.

<DefaultEventPolicy>

<StdOutMatches Operator="DoesNotMatchRegularExpression"><![CDATA[{<DataItem.+/DataItem\b\*>}|{<DataItem.\*/>}]]></StdOutMatches>

<StdErrMatches>\a+</StdErrMatches>

<ExitCodeMatches>[^0]+</ExitCodeMatches>

</DefaultEventPolicy>

EventPolicy

The EventPolicy parameter defines the expected output and the behavior based on unexpected output from the script. For the Microsoft.Windows.ScriptProbeAction module type, a System.CommandOutput data type is the expected output. This data type contains the standard output (StdOut), standard error (StdErr), and exit code of the script that ran.

The following code shows an example of the System.CommandOutput data item:

<DataItem type="System.CommandOutput" time="2008-10-10T12:48:57.0477521-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<StdOut>This is some output text</StdOut>

<StdErr>This is some error text</StdErr>

<ExitCode>0</ExitCode>

<ProcessError></ProcessError>

</DataItem>

The event policy is used to determine the behavior, depending on the output provided by the script, and is only required to be configured if you want to override the default behavior.

The script module can return one of the following four states on execution:

 Success

 Warning

 Data Loss

 Fatal Error

If this module is used by a task, the default behavior is that all states, except fatal error, are marked as being successful task execution. This means that a script could fail during execution of a task, but Operations Manager would report the task as succeeding, which can be confusing to the user. Therefore, there are situations where it may be required to make the module return a fatal error even though the script might have executed and been completed. The EventPolicy parameter controls this behavior. The EventPolicy parameter has four elements:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Severity | Integer (0 to 2) | The state of the module to return based on matching StdOut, StdErr, and exit codes.  The possible severity values and their meaning are:   0 - Warning   1 – Data Loss   2 – Fatal Error |
| StdOutMatches | PolicyExpression | Regular expression to match against data returned to standard out. |
| StdErrMatches | PolicyExpression | Regular expression to match against data returned to standard error. |
| ExitCodeMatches | PolicyExpression | Regular expression to match against the exit code from the script. |

All script-based module types have a default event policy set that is applied if no specific event policy is passed as configuration. The Microsoft.Windows.ScriptProbeAction type has the following default policy:

<DefaultEventPolicy>

<StdOutMatches/>

<StdErrMatches>\a+</StdErrMatches>

<ExitCodeMatches>[^0]+</ExitCodeMatches>

</DefaultEventPolicy>

This policy specifies that the default behavior of the module type is that any data in standard error or any non-zero exit code results in the module reporting execution as data loss (a severity of 1 is the default setting). Standard output is ignored.

When you specify an event policy setting, you are overriding the default policy. This is done per field. For example, if you override the severity only, then all other defaults will take effect.

To force the module to report a failure when standard out starts with the word “Error,” set the event policy as follows:

EventPolicy>

<Severity>2</Severity>

<StdErrMatches>^Error</StdErrMatches>

</EventPolicy>

This policy forces a module failure and reports the full output of that failure back as task output.

To always return task success and report standard output, standard error, and exit code as task output, set the policy as follows:

<EventPolicy>

<Severity>0</Severity>

<StdErrMatches>\a+</StdErrMatches>

</EventPolicy>

In the case of a rule or monitor workflow where the Microsoft.Windows.ScriptProbeAction module is used, the following behavior occurs for each non-success state:

 Warning – the workflow continues executing and uses the data item as output from the script module. Event 21413 is logged from the Health Service Modules source in the Operations Manager event log, giving full context to the warning.

 Data Loss – the data item is dropped from the workflow and no subsequent modules execute. This means that monitor state is not set for a monitor. The workflow continues to execute and is not unloaded. Event 21413 is logged from the Health Service Modules source in the Operations Manager event log, giving full context to the failure.

 Fatal Error – the module is failed, and the workflow that had the failure is unloaded and does not run again until new configuration is deployed or the agent is restarted. Event 21413 is logged, followed by Event 1103 from the HealthService source, indicating that the workflow was unloaded.

Note

The default settings for the standard error and standard output match are case insensitive and match regular expression.

There are two optional attributes that can be set on the StdOutMatches, StdErrMatches and ExitCodeMatches elements:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| Operator | PolicyOperatorType | This can be set to MatchesRegularExpression and DoesNotMatchRegularExpression. Matches is the default if this attribute is not specified. |
| CaseSensitve | Boolean | Indicates whether the match should be case-sensitive. A value of true indicates a case sensitive match only. A value of false is the default if this attribute is not specified. |

The following code shows an example event policy configuration that uses an optional attribute:

<DefaultEventPolicy>

<StdOutMatches Operator="DoesNotMatchRegularExpression"><![CDATA[<DataItem.+/DataItem\b\*>}|{<DataItem.\*/>}]]></StdOutMatches>

<StdErrMatches>\a+</StdErrMatches>

<ExitCodeMatches>[^0]+</ExitCodeMatches>

</DefaultEventPolicy>

Remarks

This module type can be used to execute scripts by specifying Cscript.exe as the command to run and passing suitable configuration. However, you should use the Microsoft.Windows.ScriptProbeAction module type instead for this purpose because it provides a pre-built composition that simplifies the parameters and is tailored for script execution scenarios.

Note

The System.CommandExecuterProbe module does not support batching. Multiple System.CommandExecuterProbe modules in a workflow will run serially, rather than in parallel.

A probe module type should not be used when any aspect of the system is being changed. There is a corresponding write action for this purpose that should be used. For more information about the write action, see the related modules section.

Examples

The following code shows a simple task that executes the IPConfig command and passes a single value to the command line:

<Task ID="Microsoft.Samples.IPConfig" Target="Windows!Microsoft.Windows.Computer" Accessibility="Internal">

<Category>Operations</Category>

<ProbeAction ID="Command" TypeID="System!System.CommandExecuterProbe">

<ApplicationName>%WINDIR%\System32\IPCONFIG.EXE</ApplicationName>

<WorkingDirectory/>

<CommandLine>/ALL</CommandLine>

<TimeoutSeconds>30</TimeoutSeconds>

<RequireOutput>true</RequireOutput>

<Files/>

</ProbeAction>

</Task>

The following code shows a more complex task example from the SQL 2005 management pack that executes a DBCC command, using a number of target properties:

<Task ID="Microsoft.SQLServer.2005.Database.DBCC.CheckAlloc" Target="SQL2005Core!Microsoft.SQLServer.2005.Database" Accessibility="Internal">

<Category>Maintenance</Category>

<ProbeAction ID="PA" TypeID="System!System.CommandExecuterProbe">

<ApplicationName>$Target/Host/Property[Type="SQL!Microsoft.SQLServer.DBEngine"]/ToolsPath$\Binn\OSQL.EXE</ApplicationName>

<WorkingDirectory>$Target/Host/Property[Type="SQL!Microsoft.SQLServer.DBEngine"]/ToolsPath$\Binn</WorkingDirectory>

<CommandLine>-S $Target/Host/Property[Type="SQL!Microsoft.SQLServer.DBEngine"]/ConnectionString$ -E -Q "DBCC CHECKALLOC('$Target/Property[Type="SQL!Microsoft.SQLServer.Database"]/DatabaseName$')"</CommandLine>

<TimeoutSeconds>300</TimeoutSeconds>

<RequireOutput>true</RequireOutput>

</ProbeAction>

</Task>

The following example shows the use of a file that is a simple PowerShell script.

<Task ID="Microsoft.Samples.PSTask" Accessibility="Internal" Target="Windows!Microsoft.Windows.Computer">

<Category>Operations</Category>

<ProbeAction ID="PS" TypeID="System!System.CommandExecuterProbe">

<ApplicationName><![CDATA[%systemroot%\System32\windowspowershell\v1.0\powershell.exe]]></ApplicationName>

<WorkingDirectory/>

<CommandLine><![CDATA[ -Command ".\GetService.ps1"]]></CommandLine>

<TimeoutSeconds>20</TimeoutSeconds>

<RequireOutput>true</RequireOutput>

<Files>

<File>

<Name>GetService.ps1</Name>

<Contents>

Get-Process

</Contents>

<Unicode>false</Unicode>

</File>

</Files>

</ProbeAction>

</Task>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| System.CommandExecuter | System.Library | Used to execute a command that changes the system in some way. This module type takes the same configuration as the probe module type that is defined in this topic. |
| System.CommandExecuterDiscoveryDataSource | System.Library | Used for a command execution that outputs discovery data. Modules of this type are used only in discovery workflows. |
| System.CommandExecuterPropertyBagSource | System.Library | Used for a command execution that outputs property bag data rather than command output. |
| Microsoft.Windows.ScriptProbeAction | Windows.Library | Used to execute a Windows Script Host script. |

For more information about executing Windows Script Host scripts and the different module types that are available, see the [Microsoft.Windows.ScriptProbeAction](#z7d0999603afd4d1889a2b779c9b5c7de) reference documentation .

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.ConsolidatorCondition

The System.ConsolidatorCondition condition detection module type is used to consolidate multiple incoming data items based on a specific schedule or a time interval. A module of this type accepts data of any type and outputs System.ConsolidatorData.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ConsolidationProperties | Complex | Contains a set of XPath expressions specifying like data items to be correlated.  This element is required but can be empty. |
| TimeXPathQuery | String | Optional. Specifies a data item property to use as the time value other than the actual time value on the data item. |
| TimeControl | Complex | Defines the time window(s) during which events will be consolidated. |
| CountingCondition | Complex | Defines the method of counting data items for consolidation. |

ConsolidationProperties

This element contains one or more XPath expressions that identify the criteria for the incoming data items to be consolidated. This is not required if an expression filter module is used (before the consolidator module) to filter the data items to a set of items that should be considered the same.

This element is always required, but it can be left empty when all incoming data items should be consolidated as one. For example:

<ConsolidationProperties/>

If there are multiple incoming data items that you want to consolidate into groups of items, then specify the data item properties that will be used to classify the data items as the same for consolidation purposes. The ConsolidationProperties element is made up of zero or more PropertyXPathQuery elements. The exact expression is dependent on the data item that is the input to the consolidation module.

For example, if you are monitoring the Windows event log, you can use the event provider data source to collect all events with event IDs in the range 101 to 110. However, you might want to output consolidated data only if there are more than 10 events of any one of these event IDs within 60 seconds. In this scenario, you specify the following configuration:

<ConsolidationProperties>

<PropertyXPathQuery>EventDisplayNumber</PropertyXPathQuery>

</ConsolidationProperties>

You can specify multiple properties that must match to consolidate the data items. The following code example shows how to use the first event parameter and the event ID to group like events together:

<ConsolidationProperties>

<PropertyXPathQuery>EventDisplayNumber</PropertyXPathQuery>

<PropertyXPathQuery>Params/Param[1]</PropertyXPathQuery>

</ConsolidationProperties>

When consolidation properties are used, the module runs all consolidation calculations in parallel with as many parallel consolidations as there are incoming groups of events. Each calculation will output data that is independent of the other consolidations in progress; subsequent modules in the workflow should take this into account.

TimeXPathQuery

By default, the time value that is used for consolidation evaluation is the time attribute of the incoming data items. It is possible to change this behavior and use a different attribute or element of the incoming data items than the time. It is not recommended that you use this parameter, and you should omit it for most monitoring scenarios.

The TimeControl parameter specifies the window that consolidation occurs in. The exact use of this window is dependent on the CountingCondition parameter that is described later in this document. There are two types of time windows available:

 Within time schedule

 Generic schedule

Note that the generic schedule cannot be used with the sliding window counting method.

The TimeControl element takes the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Latency | Integer | The time to wait after the consolidation window for data items that might be delayed. This parameter is optional, and the default is 60 seconds if it is not specified. |
| GenericSchedule | Complex | Defines a schedule of windows for consolidation to occur. A simple recurring schedule or a more complicated weekly schedule can be defined. Either GenericSchedule or WithinTimeSchedule can be defined, but not both. |
| WithinTimeSchedule | Complex | Defines a simple time window for consolidation that is triggered by incoming data items. Either GenericSchedule or WithinTimeSchedule can be defined, but not both. |

WithinTimeSchedule

This option specifies a time window in seconds. This time window is used to consolidate all data items within the window and output the consolidated data item if the configured conditions are met.

The following code example shows an example configuration of the TimeControl parameter that specifies a window of 60 seconds.

<TimeControl>

<WithinTimeSchedule>

<Interval>60</Interval>

</WithinTimeSchedule>

</TimeControl>

When this type of time control is used, the first data item starts the time window. The time window runs for the specified time. The module can output a consolidated item either during this window, at the end of the window, or not at all, depending on the counting mode configuration that is specified.

Generic Schedule

The generic schedule allows more precise window control and offers a simple recurring schedule or a weekly schedule option. Note that when a generic schedule is being used, the timing window is no longer triggered by incoming data items but is based on a set schedule.

A simple recurring schedule specifies a unit of time and a recurring interval to execute on. If the units are not specified, the value is interpreted as seconds. Optionally, a synchronization time can be specified. The following code example shows a sample generic schedule that uses the simple recurring schedule:

<GenericSchedule>

<SimpleReccuringSchedule>

<Interval>120</Interval>

</SimpleReccuringSchedule>

<ExcludeDates/>

</GenericSchedule>

To specify a value in units other than seconds, the unit attribute is used. For example:

<GenericSchedule>

<SimpleReccuringSchedule>

<Interval Unit="Minutes">30</Interval>

</SimpleReccuringSchedule>

<ExcludeDates/>

</GenericSchedule>

The allowed values for the unit attribute are:

 Seconds

 Minutes

 Hours

 Days

A synchronization time can be specified as configuration to force the window to align to a specific time of day. The synchronization time is specified in hours and minutes and uses the 24-hour format. For example:

<GenericSchedule>

<SimpleReccuringSchedule>

<Interval Unit="Minutes">30</Interval>

<SyncTime>13:00</SyncTime>

</SimpleReccuringSchedule>

<ExcludeDates/>

</GenericSchedule>

The consolidation window does not have to wait until the synchronization time for the first data consolidation. When the schedule is initialized, the next closed time is used, based on the 24-hour schedule. For example, if the preceding configuration was used, and the workflow was initialized at 13:20, the first window starts at 13:30 and then starts every 30 minutes after that point.

When a simple schedule is used, the time window is on a fixed schedule and is not affected by the incoming data items. The point at which data is output for the window is dependent on the counting mode. For the OnNewItemNOP\_OnTimerOutputRestart mode, data is output only at the end of each window if at least one data item was received as input during the window. For the other two counting modes, data can be output at any time during the window, based on the number of data items received. The following describes the behavior of each counting mode.

Either zero or one consolidated data item is output during each window for every counting mode. Data items that occur in the window after a consolidated data item has already been output are not processed.

A weekly schedule specifies a more complex set of windows for the consolidator module to evaluate. This can either be a daily schedule or a schedule that spans multiple days.

The daily schedule specifies the days of the week, the start time, and the end time. Multiple windows can be added to the configuration as required. The following example shows a window that runs from Monday to Friday from 9:00 to 10:00:

<GenericSchedule>

<WeeklySchedule>

<Windows>

<Daily>

<Start>09:00</Start>

<End>10:00</End>

<DaysOfWeekMask>62</DaysOfWeekMask>

</Daily>

</Windows>

</WeeklySchedule>

<ExcludeDates/>

</GenericSchedule>

The DaysOfWeekMask element represents one or more days of the week that the window should run. A single day or multiple days can be specified according to the following values:

|  |  |
| --- | --- |
| Day | Value |
| Sunday | 1 |
| Monday | 2 |
| Tuesday | 4 |
| Wednesday | 8 |
| Thursday | 16 |
| Friday | 32 |
| Saturday | 64 |

To specify a single day, enter the enumerator value for that day into the DaysOfWeekMask configuration element.

To specify multiple days, add the values for the days together. For example, for Monday, Wednesday and Friday you would specify 42 (2+8+32).

Multiple windows can be specified as shown in the following example, which has two windows, one from 9:00 to 10:00, Monday to Friday and the other 17:00 to 18:00, Monday and Friday:

<GenericSchedule>

<WeeklySchedule>

<Windows>

<Daily>

<Start>09:00</Start>

<End>10:00</End>

<DaysOfWeekMask>62</DaysOfWeekMask>

</Daily>

<Daily>

<Start>17:00</Start>

<End>18:00</End>

<DaysOfWeekMask>34</DaysOfWeekMask>

</Daily>

</Windows>

</WeeklySchedule>

<ExcludeDates/>

</GenericSchedule>

The daily schedule does not allow the window to span multiple days. If this is required, you must use the multiple days schedule. This schedule specifies the start and end time, together with a start and end day. The day is specified by using the day values as described previously for the DaysOfWeekMask element.

The following code example shows a window that runs from Monday at 23:00 to Tuesday at 9:00:

<GenericSchedule>

<WeeklySchedule>

<Windows>

<MultipleDays>

<Start>23:00</Start>

<StartDayOfWeekMask>2</StartDayOfWeekMask>

<End>09:00</End>

<EndDayOfWeekMask>4</EndDayOfWeekMask>

</MultipleDays>

</Windows>

</WeeklySchedule>

<ExcludeDates />

</GenericSchedule>

Two or more MultipleDays elements can be added, as required. MultipleDays elements can be combined with Daily elements, as required. For example:

<GenericSchedule>

<WeeklySchedule>

<Windows>

<MultipleDays>

<Start>23:00</Start>

<StartDayOfWeekMask>2</StartDayOfWeekMask>

<End>09:00</End>

<EndDayOfWeekMask>4</EndDayOfWeekMask>

</MultipleDays>

<MultipleDays>

<Start>23:00</Start>

<StartDayOfWeekMask>8</StartDayOfWeekMask>

<End>09:00</End>

<EndDayOfWeekMask>16</EndDayOfWeekMask>

</MultipleDays>

<Daily>

<Start>10:00</Start>

<End>11:00</End>

<DaysOfWeekMask>127</DaysOfWeekMask>

</Daily>

</Windows>

</WeeklySchedule>

<ExcludeDates />

</GenericSchedule>

Specific dates can excluded from the consolidation module when you are using either the simple recurring schedule or the weekly schedule options. Individual days are specified by using the ExcludeDates configuration.

Any number of date ranges can be added, with the constraint that a date range must be contiguous (for example, February 2 to February 4). The year is not specified when you enter the date. You can specify a single date by setting the start and end date to be the same date. The Description can be used optionally to describe why the exclusion has been made, but this does not affect the module behavior in any way.

Dates are entered in the MM/DD format.

The following code example shows two ranges specified. This stops consolidation output for the date range of between February 2 and the February 5, and also on December 1:

<ExcludeDates>

<DayInterval>

<Start>02/02</Start>

<End>02/05</End>

<Description />

</DayInterval>

<DayInterval>

<Start>12/01</Start>

<End>12/01</End>

<Description />

</DayInterval>

</ExcludeDates>

Latency

The Latency parameter specifies how long to wait after the consolidation window before either outputting data or dropping the current data. The Latency parameter is used for data items that might arrive after the time window due to a delay in delivery. For example, data items that were logged to the local Event log over a network could use the Latency parameter. If these items had arrived on time they would have fallen into the time window and their timestamp would indicate this. Therefore, the consolidator module can wait in case more items are in the window and then retroactively include them in the consolidation. The value of the Latency parameter is specified in seconds and is optional. If it is not specified, the default value is 60 seconds.

The consolidator module does not always wait for the latency period to output data. If sufficient data has arrived to meet the consolidation conditions, the module outputs when it is ready. If the count has not been reached, the module waits to ensure that no more data arrives.

For the OnNewItemNOP\_OnTimerOutputRestart counting method, there is no count value specified in the configuration. In this counting mode, the module always waits for the latency before outputting data.

CountingCondition

The CountingCondition configuration defines the method of counting incoming data items. There are three counting methods available. The behavior of each mode is compared through examples in the Remarks section of this document.

The CountingCondition element has two sub-elements:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Count | Integer | Specifies how many repeated data items are required within the counting window to trigger output. This element is not required for the OnNewItemNOP\_OnTimerOutputRestart counting method. |
| CountMode | String | Specifies the counting method. There are three values that can be specified in this parameter:   OnNewItemNOP\_OnTimerOutputRestart   OnNewItemTestOutputRestart\_OnTimerSlideByOne   OnNewItemTestOutputRestart\_OnTimerRestart |

OnNewItemNOP\_OnTimerOutputRestart

This counting method uses a fixed window that is specified in the TimeControl parameter. A consolidated data item is output only at the end of the window and never during the window. After each window either one or zero data items are output.

When this counting method is specified, no count should be configured for the module; if a count is configured, the module raises an error in the Operations Manager event log and the workflow is not run.

The following code example shows an example configuration for this mode of counting:

<CountingCondition>

<CountMode>OnNewItemNOP\_OnTimerOutputRestart</CountMode>

</CountingCondition>

This method performs no evaluation of the count of data items received in the window. If evaluation of the number of data items is required, based on the output of the consolidation, it is necessary to use an expression module after the consolidator module in the workflow.

This method is used for scenarios in which you want to count the number of events over the time window and not stop counting when you reach a threshold. This data can be mapped to an event and stored for reporting purposes, or used to drive the alerting or monitor state.

OnNewItemTestOutputRestart\_OnTimerRestart

This counting method uses a fixed window that is specified in the TimeControl parameter and a count value is configured. This count is the threshold for the number of data items that must occur during the timing window that will cause the data items in the window to be consolidated into an output from the consolidator. If this threshold is reached, the module outputs a consolidated data item immediately and does not wait for the end of the window. If the threshold is not reached and the window ends, no data is output from the consolidator. The next data item triggers a new window and disregards the events that fall into the closed window.

The following code example shows an example configuration:

<CountingCondition>

<Count>3</Count>

<CountMode>OnNewItemTestOutputRestart\_OnTimerRestart</CountMode>

</CountingCondition>

OnNewItemTestOutputRestart\_OnTimerSlideByOne

This counting method uses a sliding window that is specified in the TimeControl parameter. It is not possible to use a generic schedule with this method of counting. A simple WithinTimeSchedule configuration must be used. A count value is required for this counting method to specify the threshold for the number of data items that must occur during the timing window. With this method, the window is not fixed after it begins, and the end point can move. This method looks for a set number of data items within the specified window at whatever point the set number occurs.

When the first data item comes to the consolidator, the timing window is started. For subsequent data items, if the count threshold is reached at any point during the window, the consolidator outputs data. If the end of the window is reached, and the threshold has not been reached, the consolidator drops the first data item that came into the window, looks at the time stamp on the second data item, and resets the window to use this as the start time. This process continues until either the count threshold is reached or all data items have occurred more than one window interval in the past.

The following code example shows an example configuration for this module type:

<CountingCondition>

<Count>5</Count>

<CountMode>OnNewItemTestOutputRestart\_OnTimerSlideByOne</CountMode>

</CountingCondition>

To compare the three counting methods described in this section, see the scenario in the following Remarks section.

Remarks

The following code example shows a data item that is output from the consolidator module for Windows event data: The contents of the data item element are dependent on the type of data that is being consolidated.

<DataItem type="System.ConsolidatorData" time="2008-10-23T14:16:29.0013505-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<TimeWindowStart>2008-10-23T14:16:17.0000000-07:00</TimeWindowStart>

<TimeWindowEnd>2008-10-23T14:16:26.9999999-07:00</TimeWindowEnd>

<TimeFirst>2008-10-23T14:16:17.0000000-07:00</TimeFirst>

<TimeLast>2008-10-23T14:16:23.0000000-07:00</TimeLast>

<Count>3</Count>

<Context>

<DataItem type="Microsoft.Windows.EventData" time="2008-10-23T14:16:23.0000000-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<EventOriginId>{C9D083CE-9C36-42B9-82CE-5051C8BA180C}</EventOriginId>

<PublisherId>{D5C9D62E-11DA-4A04-242C-50418B1286A0}</PublisherId>

<PublisherName>EventCreate</PublisherName>

<EventSourceName>EventCreate</EventSourceName>

<Channel>Application</Channel>

<LoggingComputer>testmachine.example.com</LoggingComputer>

<EventNumber>206</EventNumber>

<EventCategory>0</EventCategory>

<EventLevel>4</EventLevel>

<UserName>DOMAIN\username</UserName>

<RawDescription><![CDATA[%1 ]]></RawDescription>

<LCID>1033</LCID>

<Params>

<Param>Test Event</Param>

</Params>

<EventData>

<DataItem type="System.XmlData" time="2008-10-23T14:16:23.9418446-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<EventData>

<Data>test</Data>

</EventData>

</DataItem>

</EventData>

<EventDisplayNumber>206</EventDisplayNumber>

<EventDescription><![CDATA[Test Event ]]></EventDescription>

<Keywords>36028797018963968</Keywords>

</DataItem>

</Context>

</DataItem>

In the preceding code example, consider the following sequence of events that are input to the consolidator module that begin at midnight (0:00:00):

 Event 1 – 0:00:00

 Event 2 – 0:00:08

 Event 3 – 0:00:15

 Event 4 – 0:00:17

 Event 5 – 0:00:23

For each counting method, the time control is configured to be a simple WithinTimeSchedule of 10 seconds, and there are no excludes specified. The two counting methods that take a count as configuration are set to 3.

The following consolidator output occurs for each counting method:

|  |  |
| --- | --- |
| Counting Method | Output |
| OnNewItemNOP\_OnTimerOutputRestart | Occurs at 00:00:10 after the end of the first window, which was started by event 1.  Output has a consolidated count of 2 (event 1 and 2).  Occurs at 00:00:25 after the end of the second window, which was started by event 3.  Output has a consolidated count of 3 (event 3, 4 and 5) |
| OnNewItemTestOutputRestart\_OnTimerRestart | Outputs at 00:00:23 after the third event that is received, following the start of the window that was started by event 3.  Output has a consolidated count of 3 (event 3, 4 and 5) |
| OnNewItemTestOutputRestart\_OnTimerSlideByOne | Outputs at 00:00:17 after the third event that was received, following the sliding window that was moved to start at event 2.  Output has a consolidated count of 3 (event 2, 3 and 4). |

Examples

The following rule looks for an event from the Windows application event log and consolidates, based on the sliding window counting method over a 10-second window. An alert is generated if there are more than three events received. The latency is set to 10 seconds.

<Rule ID="Microsoft.Samples.EventConsolidationTest1" Target="Microsoft.Samples.ApplicationX">

<Category>ConfigurationHealth</Category>

<DataSources>

<DataSource ID="Event" TypeID="Windows!Microsoft.Windows.EventProvider">

<ComputerName>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</ComputerName>

<LogName>Application</LogName>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>401</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<ConditionDetection ID="Consolidator" TypeID="System!System.ConsolidatorCondition">

<Consolidator>

<ConsolidationProperties/>

<TimeControl>

<Latency>10</Latency>

<WithinTimeSchedule>

<Interval>10</Interval>

</WithinTimeSchedule>

</TimeControl>

<CountingCondition>

<Count>3</Count>

<CountMode> OnNewItemTestOutputRestart\_OnTimerSlideByOne</CountMode>

</CountingCondition>

</Consolidator>

</ConditionDetection>

<WriteActions>

<WriteAction ID="GenerateAlert" TypeID="SystemHealth!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>0</Severity>

<AlertMessageId>$MPElement[Name="Microsoft.Samples.EventConsolidationTest1.AlertMessage"]$</AlertMessageId>

<Suppression/>

</WriteAction>

</WriteActions>

</Rule>

The following rule uses the non-sliding window counting method and uses a simple recurring schedule to consolidate every 5 minutes. An alert is generated if more than 10 events are received during a given window:

<Rule ID="Microsoft.Samples.EventConsolidationTest2" Target="Microsoft.Samples.ApplicationX">

<Category>ConfigurationHealth</Category>

<DataSources>

<DataSource ID="Event" TypeID="Windows!Microsoft.Windows.EventProvider">

<ComputerName>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</ComputerName>

<LogName>Application</LogName>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>401</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<ConditionDetection ID="Consolidator" TypeID="System!System.ConsolidatorCondition">

<Consolidator>

<ConsolidationProperties/>

<TimeControl>

<GenericSchedule>

<SimpleReccuringSchedule>

<Interval Unit='Minutes'>5</Interval>

<SyncTime>00:00</SyncTime>

</SimpleReccuringSchedule>

<ExcludeDates/>

</GenericSchedule>

</TimeControl>

<CountingCondition>

<Count>10</Count>

<CountMode>OnNewItemTestOutputRestart\_OnTimerRestart</CountMode>

</CountingCondition>

</Consolidator>

</ConditionDetection>

<WriteActions>

<WriteAction ID="GenerateAlert" TypeID="SystemHealth!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>0</Severity>

<AlertMessageId>$MPElement[Name="Microsoft.Samples.EventConsolidationTest2.AlertMessage"]$</AlertMessageId>

<Suppression/>

</WriteAction>

</WriteActions>

</Rule>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| System.Correlator | System.Library | Correlates a sequence of difference data items. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.Discovery.ClassSnapshotDataMapper

The System.Discovery.ClassSnapshotDataMapper is a condition detection module type that is defined in System.Library. A module of this type takes any type of data as input and outputs System.Discovery.Data data. This module type is used only in discovery workflows to map data to the required discovery data.

Parameters

The System.Discovery.ClassSnapshotDataMapper accepts the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ClassID | String | The class that is being discovered. |
| InstanceSettings | SettingsType | The property settings for the instance or instances of the class that is being discovered. |

The ClassID configuration element takes a string that must be in the $MPElement format. This element must refer to a defined class in the same management pack or a publicly available class in a referenced management pack. The $MPElement reference is validated by the MPVerify utility to ensure the reference is valid. This reference defines the class of instances that are discovered by the workflow. A single workflow that uses a module of this type can discover only a single class of instance. Other discovery workflows, such as script-based discovery can discover multiple instance types in a single workflow, but they do not use this data mapper module.

the following code example shows the ClassID configuration element.

<ClassId>$MPElement[Name="Microsoft.Samples.ApplicationX"]$</ClassId>

The InstanceSettings configuration element defines the mapping of configuration, data, target, and static values to the discovered instance or instances. The exact mapping depends on the discovery scenario and the hosting relationships that are related to the class that is being discovered.

The InstanceSettings configuration element has one or more InstanceSettings elements that contain name and value pairs. The Name element must contain a valid $MPElement reference to a class property and the Value element contains a static value, a $Data reference, a $Target reference, a $Config reference, or it contains a combination of two or more of these types.

The following code example shows a simple InstanceSettings definition:

<InstanceSettings>

<Settings>

<Setting>

<Name>$MPElement[Name="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Name>

<Value>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Value>

</Setting>

</Settings>

</InstanceSettings>

As an illustration of the use of the InstanceSettings element, consider the following scenario:

 Class A is not hosted.

 Class A hosts Class B.

 Class A has key property A1 and non-key properties A2 and A3.

 Class B hosts Class C.

 Class B has no key properties and non-key properties B1 and B2.

 Class C has key property C1 and non-key property C2.

 An instance of A and B have already been discovered.

 A discovery workflow that uses the discovery mapper module is created to discover instances of Class C and is targeted at Class B.

In this scenario the following configuration must be provided to the mapper module to discover instances of Class C:

 Property A1 of the instances of A that will host the instance of Class B, which in turn hosts the discovered Class C instances.

 Property C1 of each instance of Class C that will be discovered.

 Optionally, property C2 of each instance of Class C that will be discovered.

No properties of Class B have to be provided because this class has no key property. This means that Class A can host only a single instance. Because there is only a single instance, it is easily identified and does not have to be referenced.

The exact configuration of the mapper depends on the type of data that is input for the discovery workflow. The following code example shows a configuration that uses Windows Management Instrumentation (WMI) data as input:

<InstanceSettings>

<Settings>

<Setting>

<Name>$MPElement[Name="ClassA"]/A1$</Name>

<Value>$Target/Host/Property[Type="ClassA"]/A1$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="ClassC"]/C1$</Name>

<Value>$Data/Property[@Name=’ID’]$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="ClassC"]/C2$</Name>

<Value>$Data/Property[@Name=’Path’]$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="System!System.Entity"]/DisplayName$</Name>

<Value>$Data/Property[@Name=’Description’]$</Value>

</Setting>

</Settings>

</InstanceSettings>

A $Target reference is used for the property A1 of class A because this will be known only at run time and depends on the instances of Class A that have been discovered in your environment.

In the preceding example of the configuration, a value is specified for the DisplayName property of System.Entity. All classes inherit this property in Operations Manager. It is a best practice to set this value because it is used as the display name in the Operations console.

Remarks

This module type is used only when you are defining discovery workflows. A module of this type is not used directly in a discovery workflow because the discovery allows only a single data source module; therefore, this module type is used when you are composing a new data source module type. Many of the common scenarios for mapping data to discovery data are covered by existing data source module types, so it is rarely necessary to use this condition detection module type directly.

A module of this type can provide discovery data about a single instance or multiple instances at a time depending on the input data. Discovery data is always provided as a snapshot. Handling creation, updates, and deletions to instances is done automatically by the discovery workflow, based on changes in the snapshot of the discovery data.

To discover an instance of a class, all key properties of the discovered instances and all hosting instances must be defined in the instance settings element. Optionally, non-key properties of the discovered instance can also be defined.

Examples

The following code example is a management pack that shows the mapper module composed with a timed WMI probe, because a condition detection module cannot be used directly in a workflow. Note that this example is for illustration purposes only. For timed WMI discovery you should use the predefined Microsoft.Windows.WmiProviderWithClassSnapshotDataMapper module type. The management pack discovers shared folders on the computer and creates an object of type Microsoft.Samples.SharedFolder for each instance that is found.

<ManagementPack xsi:noNamespaceSchemaLocation="C:\MPSchema\ManagementPackSchema.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Manifest>

<Identity>

<ID>Microsoft.Samples.Discovery</ID>

<Version>1.0.0.0</Version>

</Identity>

<Name>Microsoft.Samples.Discovery</Name>

<References>

<Reference Alias="System">

<ID>System.Library</ID>

<Version>6.0.4941.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="Windows">

<ID>Microsoft.Windows.Library</ID>

<Version>6.0.4941.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

</References>

</Manifest>

<TypeDefinitions>

<EntityTypes>

<ClassTypes>

<ClassType ID="Microsoft.Samples.SharedFolder" Abstract="false" Accessibility="Internal" Hosted="true" Base="System!System.LogicalEntity">

<Property ID="ShareName" Key="true" Type="string"/>

<Property ID="Path" Type="string"/>

<Property ID="Description" Type="string"/>

</ClassType>

</ClassTypes>

<RelationshipTypes>

<RelationshipType ID="Microsoft.Samples.WindowsComputerHostsSharedFolder" Abstract="false" Accessibility="Internal" Base="System!System.Hosting">

<Source>Windows!Microsoft.Windows.Computer</Source>

<Target>Microsoft.Samples.SharedFolder</Target>

</RelationshipType>

</RelationshipTypes>

</EntityTypes>

<ModuleTypes>

<DataSourceModuleType ID="Microsoft.Samples.TimedWMIDiscovery" Accessibility="Internal">

<Configuration>

<IncludeSchemaTypes>

<SchemaType>System!System.Discovery.MapperSchema</SchemaType>

</IncludeSchemaTypes>

<xsd:element name="NameSpace" type="xsd:string"/>

<xsd:element name="Query" type="xsd:string"/>

<xsd:element name="Frequency" type="xsd:unsignedInt"/>

<xsd:element name="ClassId" type="xsd:string"/>

<xsd:element name="InstanceSettings" type="SettingsType"/>

</Configuration>

<ModuleImplementation>

<Composite>

<MemberModules>

<DataSource ID="WMISource" TypeID="Windows!Microsoft.Windows.WmiProvider">

<NameSpace>$Config/NameSpace$</NameSpace>

<Query>$Config/Query$</Query>

<Frequency>$Config/Frequency$</Frequency>

</DataSource>

<ConditionDetection ID="Mapper" TypeID="System!System.Discovery.ClassSnapshotDataMapper">

<ClassId>$Config/ClassId$</ClassId>

<InstanceSettings>$Config/InstanceSettings$</InstanceSettings>

</ConditionDetection>

</MemberModules>

<Composition>

<Node ID="Mapper">

<Node ID="WMISource"/>

</Node>

</Composition>

</Composite>

</ModuleImplementation>

<OutputType>System!System.Discovery.Data</OutputType>

</DataSourceModuleType>

</ModuleTypes>

</TypeDefinitions>

<Monitoring>

<Discoveries>

<Discovery ID="Microsoft.Samples.SharedFolder.Discovery" Target="Windows!Microsoft.Windows.Server.Computer" Remotable="false" Enabled="true">

<Category>Discovery</Category>

<DiscoveryTypes>

<DiscoveryClass TypeID="Microsoft.Samples.SharedFolder">

<Property PropertyID="ShareName"/>

<Property PropertyID="Path"/>

<Property PropertyID="Description"/>

<Property TypeID="System!System.Entity" PropertyID="DisplayName"/>

</DiscoveryClass>

</DiscoveryTypes>

<DataSource ID="DS" TypeID="Microsoft.Samples.TimedWMIDiscovery">

<NameSpace>\\$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$\root\cimv2</NameSpace>

<Query>Select \* from Win32\_Share</Query>

<Frequency>120</Frequency>

<ClassId>$MPElement[Name="Microsoft.Samples.SharedFolder"]$</ClassId>

<InstanceSettings>

<Settings>

<Setting>

<Name>$MPElement[Name="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Name>

<Value>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="Microsoft.Samples.SharedFolder"]/ShareName$</Name>

<Value>$Data/Property[@Name='Name']$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="Microsoft.Samples.SharedFolder"]/Path$</Name>

<Value>$Data/Property[@Name='Path']$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="Microsoft.Samples.SharedFolder"]/Description$</Name>

<Value>$Data/Property[@Name='Description']$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="System!System.Entity"]/DisplayName$</Name>

<Value>$Data/Property[@Name='Path']$</Value>

</Setting>

</Settings>

</InstanceSettings>

</DataSource>

</Discovery>

</Discoveries>

</Monitoring>

<LanguagePacks>

<LanguagePack ID="ENU" IsDefault="true">

<DisplayStrings>

<DisplayString ElementID="Microsoft.Samples.Discovery">

<Name>Micorosft Samples - Discovery</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.Samples.SharedFolder">

<Name>Shared Folder</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.Samples.SharedFolder" SubElementID="ShareName">

<Name>Name</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.Samples.SharedFolder" SubElementID="Path">

<Name>Path</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.Samples.SharedFolder" SubElementID="Description">

<Name>Description</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.Samples.SharedFolder.Discovery">

<Name>Discovery shared folders</Name>

</DisplayString>

</DisplayStrings>

</LanguagePack>

</LanguagePacks>

</ManagementPack>

Related Modules

Usually you are not required to compose your own modules by using the System.Discovery.ClassSnapshotDataMapper module type because many of the common workflows that are required for discoveries are already defined in the system libraries. However, this module is often used in the predefined discovery modules, which means that the ClassId and InstanceSettings configuration elements must be configured in the same way that is described in this document. The following table summarizes other related module types.

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.RegistryDiscoveryProvider | Windows.Library | Timed registry discovery |
| Microsoft.Windows.Win32ServiceInformationProviderWithClassSnapshotDataMapper | Windows.Library | Timed Windows service discovery |
| Microsoft.Windows.WmiProviderWithClassSnapshotDataMapper | Windows.Library | Timed WMI discovery |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.Event.GenericDataMapper

The System.Event.GenericDataMapper is a condition-detection module type that is used to map data of any type to event data that can be stored in the Operations Manager database and the data warehouse. Only event data, performance data, state change data, and alert data can be stored natively in the databases, so it might be necessary to map data to a supported type.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| EventOriginId | String | Identifies the source of the event by a GUID. The value of this parameter is commonly set to a $MPElement$ variable that contains a GUID that represents the ID of the rule that is running. |
| PublisherId | String | Identifies the event publisher by a GUID. This parameter is commonly set to a $MPElement$ variable that contains a GUID that represents the ID of the rule running. |
| PublisherName | String | Contains the event source. |
| Channel | String | Contains the name of the event log. |
| LoggingComputer | String | Contains the name of the computer the event was logged from. |
| EventNumber | String | Contains the event ID. |
| EventCategory | String | Contains the category of the event that is used to help correlate events to the components of an application. |
| EventLevel | String | Contains the severity level of the event (for example, Information, Warning, or Error). |
| Description | String | Contains the full-text description of the event. This string can optionally contain a parameter substitution (see comments below). |
| UserName | String | Contains the username of the user that generated the event. |
| Params | ParamListType | Optional. Specifies one of more event parameters. |

No parameters are marked as overrideable for this module type.

PublisherName

The PublisherName parameter contains the event source of the event that would be displayed in the Event Viewer if this was an actual generated Windows event. This value can be any string and have to be a registered event source. For example:

<PublisherName>ApplicationX</PublisherName>

Note

It is a best practice to use a publisher name that can easily be attributed to the application you are monitoring.

Channel

The Channel parameter contains the Windows event log name that the event appears to originate from, such as the Application or System event log. This value does not have to be a registered event log on the Windows-based computer because the event is being generated only for Operations Manager use.

The following code is an example definition:

<Channel>Application</Channel>

LoggingComputer

The LoggingComputer parameter specifies the computer that the event appears to be generated from. This is usually the computer on which the workflow is executing, and it is commonly a $Target reference.

For example, if the module is being used in a rule that is targeting an instance that is hosted by Windows-based computer, the configuration of this parameter is defined as the following:

<LoggingComputer>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</LoggingComputer>

EventNumber

The EventNumber parameter contains the ID of the event. Although the schema specifies a string for this parameter, the value should be an integer.

The following code example shows the use of a static value:

<EventNumber>200</EventNumber>

The following code example shows the use of a value from the incoming data item, which is a property bag in this specific example:

<EventNumber>$Data/Property[@Name='EventIDValue']$</EventNumber>

EventCategory

The value of the EventCategory parameter (an integer from 0 to 65535) is an index into a category DLL message table that contains a localized string. Each publisher defines its own set of categories. These categories commonly correspond to individual components (for example: a connector, module host, or data warehouse).

This value can also be set to 0 and ignored.

EventLevel

The EventLevel parameter specifies the severity level of the event. The following values are valid:

|  |  |
| --- | --- |
| Value | Definition |
| 0 | Success |
| 1 | Critical |
| 2 | Warning |
| 4 | Information |
| 8 | Success Audit |
| 16 | Failure Audit |

UserName

The UserName parameter contains the user name of the user that the event appears to be generated from. This element must be present, but it can be empty.

Description

The Description parameter contains the full text description. This value can optionally use parameter substitution from the Params section of the configuration. The full substituted string is stored in the database and visible in the Operations console.

Important

This text is not localizable.

The following code shows an example static description:

<Description>Application X has a problem</Description>

The following code shows an example that uses two parameters that are substituted into the description:

<Description>Value1 is: %1; Value 2 is %2</Description>

This example requires that you set a minimum of two parameters in the Params configuration value for the mapper module. If you specify a parameter in the description and it is not populated in the parameters element, the description is empty for that parameter.

For more detailed information about escape sequences and parameter substitution, see the [documentation for the Windows SDK FormatMessage function](http://go.microsoft.com/fwlink/?LinkId=137113).

Param

The Param parameter contains the set of event parameters used to simulate a parameterized event. These parameters can be used to build the description of the event if required. This element is optional, but if it is used it should contain at least one Param element.

You can use static values, $Data values, and $Target values, as required, in the parameters.

The following code is an example setting three parameters:

<Params>

<Param>$Data/Property[@Name='Value1']$</Param>

<Param>$Data/Property[@Name='Value2']$</Param>

<Param>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</Param>

</Params>

Remarks

It is only necessary to map data to event data if you want to store the data in the database as an event. In monitoring scenarios you can use any data type to trigger a state change on a monitor or generate an alert. The raw data item is stored as the context of the state change or alert, and it appears in the Operations console.

Because it is only necessary to store data as events if you want to report on or view the data as events, the event mapper is most commonly used in collection rules.

The events that are created with the mapper module never appear in the Windows event log. These are simulated events that are generated and stored only in the Operations Manager database or the data warehouse database or both.

Any data reference that is used in the mapper configuration is dependent on the data type of the input to the mapper module.

Examples

The following code example shows a rule that executes a script on a timed interval. The data in the property bag output from the script is used to generate an event and store it in the Operations Manager database. The rule uses a combination of static values and data item values to create an event with Event ID 101 from source Application X.

For this specific scenario you should use the Microsoft.Windows.TimedScript.EventProvider that is already defined in the Microsoft.Windows.Library, which provides the composition of the script and the mapper in a data source.

The following code example is shown for illustration purposes only.

<Rule ID="Microsoft.Samples.EventMapper" Target="Microsoft.Samples.ApplicationX" Enabled="false">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Script" TypeID="Windows!Microsoft.Windows.TimedScript.PropertyBagProvider">

<IntervalSeconds>60</IntervalSeconds>

<SyncTime/>

<ScriptName>Microsoft.Samples.EventMapper.vbs</ScriptName>

<Arguments/>

<ScriptBody>

<![CDATA[

Option Explicit

Dim OMApi, OMPBag

Set OMApi = CreateObject("MOM.ScriptAPI")

Set OMPBag = OMApi.CreatePropertyBag()

Call OMPBag.AddValue("Value1", "Test1")

Call OMPBag.AddValue("Value2", "Test2")

Call OMPBag.AddValue("Value3", 101)

Call OMAPI.Return(OMPBag)

]]>

</ScriptBody>

<TimeoutSeconds>30</TimeoutSeconds>

</DataSource>

</DataSources>

<ConditionDetection ID="Mapper" TypeID="System!System.Event.GenericDataMapper">

<EventOriginId>$MPElement$</EventOriginId>

<PublisherId>$MPElement$</PublisherId>

<PublisherName>ApplicationX</PublisherName>

<Channel>Application</Channel>

<LoggingComputer>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</LoggingComputer>

<EventNumber>$Data/Property[@Name='Value3']$</EventNumber>

<EventCategory>0</EventCategory>

<EventLevel>4</EventLevel>

<UserName/>

<Description>Value1 is: %1; Value 2 is %2</Description>

<Params>

<Param>$Data/Property[@Name='Value1']$</Param>

<Param>$Data/Property[@Name='Value2']$</Param>

</Params>

</ConditionDetection>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

</WriteActions>

</Rule>

The exact referencing format to obtain values from the incoming data item is dependent on the type of data being mapped. The preceding example maps property bag data (System.PropertyBagData) as the input to the mapper.

The following code is an example of the property bag data item output by the script data source:

<DataItem type="System.PropertyBagData" time="2008-10-16T17:54:48.0580031-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<Property Name="Value1" VariantType="8">Test1</Property>

<Property Name="Value2" VariantType="8">Test2</Property>

<Property Name="Value3" VariantType="2">101</Property>

</DataItem>

In the preceding example, this data is mapped to the following event data item and stored in the database:

<DataItem type="System.Event.Data" time="2008-10-16T18:02:50.0421967-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<EventOriginId>{0488804B-8468-7A9A-143B-ED7CC7317E4A}</EventOriginId>

<PublisherId>{7A495B8A-0DC7-7139-D9A2-0DC0A2481868}</PublisherId>

<PublisherName>ApplicationX</PublisherName>

<EventSourceName>ApplicationX</EventSourceName>

<Channel>Application</Channel>

<LoggingComputer>stwilsonx20.redmond.corp.microsoft.com</LoggingComputer>

<EventNumber>101</EventNumber>

<EventCategory>0</EventCategory>

<EventLevel>16</EventLevel>

<UserName></UserName>

<RawDescription>Value1 is: %1; Value 2 is %2</RawDescription>

<CollectDescription Type="Boolean">true</CollectDescription>

<Params>

<Param>Test1</Param>

<Param>Test2</Param>

</Params>

<EventData>

<DataItem type="System.PropertyBagData" time="2008-10-16T18:02:50.0421967-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<Property Name="Value1" VariantType="8">Test1</Property>

<Property Name="Value2" VariantType="8">Test2</Property>

<Property Name="Value3" VariantType="2">101</Property>

</DataItem>

</EventData>

<EventDisplayNumber>101</EventDisplayNumber>

<EventDescription>Value1 is: Test1; Value 2 is Test2</EventDescription>

</DataItem>

Certain data source modules can output multiple data items at a time. If multiple data items are passed to a mapper module, multiple events are created.

The following example shows a rule that executes a script on a timed interval. In this script three property bags are created. This results in three events being created with different event IDs (102, 103 and 104) from event source Application X and stored in the Operations Manager database:

<Rule ID="Microsoft.Samples.MultiEventMapper" Target="Microsoft.Samples.ApplicationX" Enabled="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Script" TypeID="Windows!Microsoft.Windows.TimedScript.PropertyBagProvider">

<IntervalSeconds>60</IntervalSeconds>

<SyncTime/>

<ScriptName>Microsoft.Samples.MultiEventMapper.vbs</ScriptName>

<Arguments/>

<ScriptBody>

<![CDATA[

Option Explicit

Dim OMApi, OMPBag

Set OMApi = CreateObject("MOM.ScriptAPI")

Randomize()

Dim i

For i = 1 to 3

Set OMPBag = OMApi.CreatePropertyBag()

Call OMPBag.AddValue("Value1", "Test1")

Call OMPBag.AddValue("Value2", "Test2")

Call OMPBag.AddValue("Value3", 101 + i)

Call OMApi.AddItem(OMPBag)

Next

Call OMAPI.ReturnItems()

]]>

</ScriptBody>

<TimeoutSeconds>30</TimeoutSeconds>

</DataSource>

</DataSources>

<ConditionDetection ID="Mapper" TypeID="System!System.Event.GenericDataMapper">

<EventOriginId>$MPElement$</EventOriginId>

<PublisherId>$MPElement$</PublisherId>

<PublisherName>ApplicationX</PublisherName>

<Channel>Application</Channel>

<LoggingComputer>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</LoggingComputer>

<EventNumber>$Data/Property[@Name='Value3']$</EventNumber>

<EventCategory>0</EventCategory>

<EventLevel>16</EventLevel>

<UserName/>

<Description>Value1 is: %1; Value 2 is %2</Description>

<Params>

<Param>$Data/Property[@Name='Value1']$</Param>

<Param>$Data/Property[@Name='Value2']$</Param>

</Params>

</ConditionDetection>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

</WriteActions>

</Rule>

Related Modules

There are many module types already available that combine a data source with an event mapper module in their composition. You should use these prebuilt module types where applicable. The configuration on these module types exposes either the full System.Event.GenericDataMapper configuration schema described in this article or a subset of the schema.

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.TimedScript.EventProvider | Windows.Library | Generates an event from a timed script. |
| Microsoft.Windows.WmiEventProvider.EventProvider | Windows.Library | Generates an event from a Windows Management Instrumentation (WMI) event. |
| Microsoft.Windows.WmiEventProvider.FilteredEventProvider | Windows.Library | Generates an event from a filtered WMI event. |
| Microsoft.Windows.WmiProvider.EventProvider | Windows.Library | Generates an event from a WMI query. |
| Microsoft.Windows.WmiProvider.FilteredEventProvider | Windows.Library | Generates an event from a filtered WMI query. |
| System.ApplicationLog.GenericCSVLog.EventProvider | System.ApplicationLog.Library | Generates an event from a CSV text log file. |
| System.ApplicationLog.GenericCSVLog.FilteredEventProvider | System.ApplicationLog.Library | Generates an event from a filtered CSV text log file. |
| System.ApplicationLog.GenericLog.EventProvider | System.ApplicationLog.Library | Generates an event from a text log file. |
| System.ApplicationLog.GenericLog.FilteredEventProvider | System.ApplicationLog.Library | Generates an event from a filtered text log file. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.ExpressionFilter

The System.ExpressionFilter condition detection module is a filtering module that takes any input data type and outputs the same data type. There are no changes to the data item flowing through a module of this type. The module either allows the data item through or stops the data item based on the expression specified as configuration.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Expression | ExpressionType | Contains the definition of an expression to evaluate as true or false. |

A module of this type takes a single Expression element to specify the filter conditions. The expression consists of one or more comparison expressions using logical comparison operations (AND, OR and NOT). The entire expression is evaluated either to true or to false. If the entire expression evaluates to true, the data item is passed to subsequent modules in the workflow. If the entire expression evaluates to false, the data item is not passed to subsequent modules in the workflow.

The ExpressionType schema allows for many different comparison scenarios.

There are two basic types of expressions that can be combined with logical operators:

|  |  |
| --- | --- |
| Expression Type | Description |
| SimpleExpression | Evaluates two values by using a simple comparison (equal, not equal, less than, greater than, less than or equal, greater than or equal). The two values can be cast to the appropriate type for comparison. |
| RegexExpression | Evaluates one value against a regular expression or a wildcard expression; or searches for a substring. |

SimpleExpression

A SimpleExpression type compares two values (left side and right side) with a simple operator of the following types:

|  |  |
| --- | --- |
| Operator | Description |
| NotEqual | Evaluates to true if the left side of the expression is not equal to the right side. This evaluation is case-insensitive by default. |
| Greater | Evaluates to true if the left side of the expression is greater in value than the right side. This is a string comparison by default. |
| Less | Evaluates to true if the left side of the expression is less in value than the right side. This is a string comparison by default. |
| GreaterEqual | Evaluates to true if the left side of the expression is greater than or equal in value than the right side. This is a string comparison by default. |
| LessEqual | Evaluates to true if the left side of the expression is less than or equal in value than the right side. This is a string comparison by default. |

The expression can be specified as a static value, a reference to a property of the incoming data item, a target reference, or a configuration reference. The expression is specified by a left side and right side together with a logical operator. The simple expression evaluates to true or false based on the configuration. The expression is defined by using a ValueExpression element. Within that element, you can use either an XPathQuery element to reference a value in the incoming data element or a Value element to reference a static, $Target, or $Config reference.

Note

Specifying XPathQuery as the value expression type implicitly configures the module to look inside the data item, so the $Data reference is not required. This is the only module type that does not require $Data syntax.

The following code is a simple example that compares the PublisherName of the incoming data item (a Windows event, in this example) to a static value of EventCreate:

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

For the comparison, the first value expression that is listed in the configuration is the left side and the second value expression is the right side of the logical expression.

It is valid to use XPathQuery or Value expressions on the left side or right side of the comparison, as required. For example, the preceding example could also be written as:

<Expression>

<SimpleExpression>

<ValueExpression>

<Value>EventCreate</Value>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<XPathQuery>PublisherName</XPathQuery>

</ValueExpression>

</SimpleExpression>

</Expression>

It is also valid to compare two Value expressions or two XPath expressions. The expression module performs a string comparison of the two values that are specified by default.

The expression module always compares the left side and right side as strings unless otherwise specified.

It is possible to cast the value expressions to an appropriate type for comparison. The allowed type casts are:

 Boolean

 Integer

 UnsignedInteger

 Double

 Duration

 DateTime

 String

The following example forces an Integer comparison:

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="Integer">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>LessEqual</Operator>

<ValueExpression>

<Value Type="Integer">102</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

If the value cannot be converted to the requested type, an error event that describes the problem is logged to the local Operations Manager event log. The left side and right side of the expression should be cast to the same value type, or it might not be possible to evaluate the expression.

When you compare string values, you can specify that the comparison is case sensitive. The default is a case-insensitive match unless otherwise specified. To force a case-sensitive comparison, use the CaseSensitive attribute on the operator:

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator CaseSensitive="true">Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

RegexExpression

The RegExExpression type compares a value expression (left side) to a wildcard or regular expression (right side). The value expression is specified in the same way as the simple expression described previously. The regular expression is specified by using a pattern element.

The allowed operators for regular expression comparisons are:

|  |  |
| --- | --- |
| Operator | Description |
| MatchesWildcard | The expression evaluates as true if it matches the specified wildcard expression. The following wildcard characters are allowed:   # - represents a number [0-9].   ? – represents any single character.   \* - represents any sequence of characters.   \ - escapes the following character; for example, \# is used to specify the # character. |
| ContainsSubstring | Expression evaluates as true if it contains the substring specified in the pattern. The pattern is mapped to a regular expression such as: "^.\*EscapedPattern.\*$". |
| MatchesRegularExpression | The expression evaluates as true if it matches the specified regular expression pattern. The regular expression evaluation uses the ATL regular expression engine. |
| DoesNotContainSubstring | The expression evaluates as true if it does not contain the substring specified in the pattern. |
| DoesNotMatchWildcard | The expression evaluates as true if it does not match the specified wildcard pattern. |
| DoesNotMatchRegularExpression | The expression evaluates as true if it does not match the specified regular expression pattern. |

Note

Some Operations Manager 2005 comparison types are allowed in the operator for backward compatibility only. These should not be used for new management packs.

The following code shows an example of the use of the RegExExpression:

<Expression>

<RegExExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>MatchesRegularExpression</Operator>

<Pattern>^(200|201|202|203)$</Pattern>

</RegExExpression>

</Expression>

The two expression types in the previous example can be combined to form multiple comparisons. For example, it would not be possible to compare the event ID and publisher name in an event data item in a single expression. Multiple expression types can be combined by using the following logical operators between expression definitions:

|  |  |
| --- | --- |
| Operator Element | Description |
| And | Combines two or more expressions with a logical AND. All expressions contained in the And element must evaluate to true for the overall expression to evaluate to true; otherwise the result is false. |
| Or | Combines two or more expressions with a logical OR. At least one expression contained in the Or element must evaluate to true for the overall expression to evaluate to true; otherwise the result is false. |
| Not | Contains only a single expression. This element reverses the result of the contained expression, so if the contained expression returned true the overall result would be false. |

Multiple expression types are represented in the XML configuration with the operator as the element and then one or more expressions contained as sub-elements.

The following example shows how to combine two expressions in an AND operation:

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">101</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

There is no limit to how many expressions can be combined with a single operator. Multiple operators can be combined as required.

The following code shows example shows an AND expression contained within an OR expression:

<Expression>

<Or>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">101</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<RegExExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>MatchesRegularExpression</Operator>

<Pattern>^(200|201|202|203)$</Pattern>

</RegExExpression>

</Expression>

</And>

</Expression>

</Or>

</Expression>

The following code shows how the preceding example could be written logically:

(PublisherName = EventCreate AND EventDisplayNumber = 101) OR (PublisherName = EventCreate2 AND EventDisplayNumber Matches ^(200|201|202|203)$)

Remarks

The System.ExpressionFilter module is used in many composite module type definitions. For any module configuration that takes an element of type ExpressionType, the configuration will be the same format as what is described in this topic.

Examples

The following example shows a collection using a simple expression:

<Rule ID="Microsoft.Samples.CollectEvent101" Target="Microsoft.Samples.ApplicationX">

<Category>AvailabilityHealth</Category>

<DataSources>

<DataSource ID="DS" TypeID="Windows!Microsoft.Windows.EventProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<LogName>Application</LogName>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">101</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">Params/Param[1]</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">$Target/Property[Type="Microsoft.Samples.ApplicationX"]/ServiceName$</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishEventData"/>

</WriteActions>

</Rule>

The following example shows a rule that generates an alert if the specified registry key does not exist:

<Rule ID="Microsoft.Samples.RegKeyTest" Target="Microsoft.Samples.ApplicationX">

<Category>Custom</Category>

<DataSources>

<DataSource ID="DS" TypeID="Windows!Microsoft.Windows.RegistryProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<RegistryAttributeDefinitions>

<RegistryAttributeDefinition>

<AttributeName>ApplicationXExists</AttributeName>

<Path>SOFTWARE\MicrosoftModuleSample\ApplicationX</Path>

<PathType>0</PathType>

<AttributeType>0</AttributeType>

</RegistryAttributeDefinition>

</RegistryAttributeDefinitions>

<Frequency>120</Frequency>

</DataSource>

</DataSources>

<ConditionDetection ID="Filter" TypeID="System!System.ExpressionFilter">

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="Boolean">Values/ApplicationXExists</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="Boolean">false</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</ConditionDetection>

<WriteActions>

<WriteAction ID="GenerateAlert" TypeID="SystemHealth!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>1</Severity>

<AlertMessageId>$MPElement[Name="Microsoft.Samples.RegKeyTest.AlertMessage"]$</AlertMessageId>

<AlertParameters>

<AlertParameter1>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</AlertParameter1>

</AlertParameters>

<Suppression>

<SuppressionValue/>

</Suppression>

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.Health.GenerateAlert

The System.Health.GenerateAlert is a write action module type that accepts any type of data. A module of this type is used in a rule to generate an alert.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Priority | Integer (0 to 2) | Specifies the priority (low, medium, high) for the alert. |
| Severity | Integer (0 to 2) | Contains severity (information, warning, error) for the alert. |
| AlertName | String | Deprecated – use AlertMessageId instead. |
| AlertDescription | String | Deprecated – use AlertMessageId instead. |
| AlertOwner | String | Contains the owner for the alert. This parameter sets the initial owner, but it can be changed by a user when the alert is generated. |
| AlertMessageId | String | Contains up to 10 alert parameters that are used to populate the alert description. |
| AlertParameters | System.Health.AlertParameters | Contains up to 10 alert parameters that are used to populate the alert description. |
| Suppression | System.Health.Suppression | Defines how multiple alerts from the same rule are suppressed. |
| Custom1 | String | Specifies the value for custom alert field 1. |
| Custom2 | String | Specifies the value for custom alert field 2. |
| Custom3 | String | Specifies the value for custom alert field 3. |
| Custom4 | String | Specifies the value for custom alert field 4. |
| Custom5 | String | Specifies the value for custom alert field 5. |
| Custom6 | String | Specifies the value for custom alert field 6. |
| Custom7 | String | Specifies the value for custom alert field 7. |
| Custom8 | String | Specifies the value for custom alert field 8. |
| Custom9 | String | Specifies the value for custom alert field 9. |
| Custom10 | String | Specifies the value for custom alert field 10. |

All parameters are optional, except Priority and Severity. However, either AlertName or AlertMessageId must be specified. Priority and Severity are the only fields that are defined as overridable in the module type.

Severity

Severity defines the alert severity and can be one of the following values:

 0 - Information

 1 - Warning

 2 – Critical

Priority

Priority defines the alert priority and the mapping, and it can be one of the following values:

 0 - Low

 1 - Medium

 2 – High

The default Active Alerts view in the Operations console does not show the priority of the alert, but you can add this column by personalizing the view if required.

Alert Owner

The AlertOwner parameter is used to set an owner for the alert. The meaning and possible values for this field are specific to a customer environment. It can be set at authoring time as required, but often it is omitted, especially for management packs that are shipped to other customers. Alert owner is an available property on the generated alert that can be set using the Operations console after an alert is generated.

AlertMessageID

The AlertMessageId parameter references a valid string resource element in the same management pack in which the rule that is using this module type is defined. The string resource element is used so that an alert name and description can be assigned to the alert in a localizable manner. The following code shows an example configuration:

<AlertMessageId>$MPElement[Name="AuthorMPs.ModuleSamples.ApplicationX.AlertOnEvent101.AlertMessage"]$</AlertMessageId>

You must use the $MPElement reference syntax to refer to a valid string resource. Verification of the management pack fails if the string resource does not exist in the management pack. The corresponding string resource is defined in the Presentation section of the management pack as:

<StringResources>

<StringResource ID="AuthorMPs.ModuleSamples.ApplicationX.AlertOnEvent101.AlertMessage"/>

</StringResources>

AlertParameters

The AlertParameters element is an optional element that consists of between one to 10 alert parameter elements. Alert parameters are used to build an alert description that uses $Data and $Target values. If the alert description contains only static data, this element is not required. The alert parameters are used with the string resource that is referenced in the AlertMessageId to create the alert text. An following example shows a set of parameters:

<AlertParameters>

<AlertParameter1>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</AlertParameter1>

<AlertParameter2>$Data/EventDisplayNumber$</AlertParameter2>

</AlertParameters>

The following example is for a rule that is targeted at an application hosted by a Windows-based computer and monitors the Windows event log. This set of alert parameters provides the name of the computer athat the alert is generated on and the event ID of the event that generated the alert. These parameters can be used to build the alert description by creating a display string for the string resource that was referenced in the AlertMessageId element of the module configuration.

<DisplayString ElementID="Microsoft.Samples.ApplicationX.AlertOnEvent101.AlertMessage">

<Name>Application X Failure</Name>

<Description>Critical failure in Application X running on computer {0}. Event received was event ID: {1}</Description>

</DisplayString>

Substitution place holders are used to refer to the alert parameters that are defined in the module configuration. Alert parameter 1 replaces {0}; alert parameter 2 l replaces {1}. As many as ten alert parameters can be defined in the module configuration. The preceding example might result in a description such as:

Critical failure in Application X running on computer computer001.mydomain.com. Event received was event ID: 101

Alert parameters can be used only in the alert description, and cannot be used in the alert name.

Suppression

The Suppression element is used to control how alerts are suppressed, based on repeater trigger conditions. There are three possible options:

 Do not suppress alerts and create a new alert for every occurrence of the trigger event.

 Suppress to one alert per workflow instance; this creates a maximum of one alert for each monitored object of the class that the rule is targeted at. A new alert is not created until the open alert is resolved.

 Suppress based on values in the data that trigger the alert; for example, event parameters.

To configure the module with no suppression, use an empty element. For example:

<Suppression/>

To suppress on workflow, the suppression element contains an empty SuppressionValue element as follows:

<Suppression>

<SuppressionValue/>

</Suppression>

To suppress on some value in the data item that is passed to the module, you can specify one or more $Data references as suppression values. In the following example, a Windows event alert generating rule is configured to suppress alerts only if the first event parameter matches the previous data item that generated an alert:

<Suppression>

<SuppressionValue>$Data/Params/Param[1]$</SuppressionValue>

</Suppression>

Custom Parameters

The custom parameters are used to store additional information in the alert properties. This information appears in the Operations console and can be used to store any data. The data can be static, or it can contain $Target and $Data references and be populated at run time.

The following example shows a configuration for Custom1:

<Custom1>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</Custom1>

Custom alert parameters are often used when you are coding alerts for forwarding to a ticketing system, and they are implemented either as part of the rule definition or after the alert is received by an operator.

Remarks

The preferred way to generate alerts is by using monitors that monitor the state of a specific aspect of an application or component and an alert based on state changes. However, when no state can be inferred from a system event, alert generating rules can be used.

When repeat alerts are suppressed, the repeat count on the alert is incremented and can be seen in the alert properties page of the Operations console. Only the first data item that triggered the alert is saved as context, and further data items are not stored.

Suppression is in place only while the alert is open. After the alert is closed, a new alert is opened by the first event that matches the trigger conditions.

Examples

The following example shows an alert-generated rule that generates an alert when an event ID 101 is received from source EventCreate. The alert is suppressed on workflow. Two alert parameters are defined, and one custom field is set.

<Rule ID="Microsoft.Samples.AlertOnEvent101" Target="Microsoft.Samples.ApplicationX" Enabled="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="DS" TypeID="Windows!Microsoft.Windows.EventProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<LogName>Application</LogName>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">PublisherName</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">EventCreate</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">EventDisplayNumber</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">101</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="GenerateAlert" TypeID="SystemHealth!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>0</Severity>

<AlertMessageId>$MPElement[Name="Microsoft.Samples.AlertOnEvent101.AlertMessage"]$</AlertMessageId>

<AlertParameters>

<AlertParameter1>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</AlertParameter1>

<AlertParameter2>$Data/EventDisplayNumber$</AlertParameter2>

</AlertParameters>

<Suppression>

<SuppressionValue/>

</Suppression>

<Custom1>$Target/Host/Property[Type='Windows!Microsoft.Windows.Computer']/NetworkName$</Custom1>

</WriteAction>

</WriteActions>

</Rule>

The StringResource element is defined as:

<StringResource ID="Microsoft.Samples.AlertOnEvent101.AlertMessage"/>

The display string for the StringResource element is defined as:

<DisplayString ElementID="AuthorMPs.Samples.AlertOnEvent101.AlertMessage ">

<Name>Application X Failure</Name>

<Description>Critical failure in Application X running on computer {0}. Event received was event ID: {1}</Description>

</DisplayString>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.Performance.AveragerCondition

The System.Performance.AverageCondition module is a condition detection module type that accepts System.Performance.Data as the input data type and outputs System.Performance.Data. A module of this type is used to average a number of performance samples and output average values instead of the raw input values.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| NumSamples | Unsigned Integer | Contains the number of samples to average over. |

Remarks

A module of this type generates data at the same frequency as the input data items after the initial number of specified samples have passed. For example, if the frequency of the incoming data is once every 60 seconds and the module is configured to sample over 4 samples, there will be no data item output for 240 seconds after the workflow is initialized for the first time.

Examples

The following example shows a simple performance collection rule that reads data from a Windows performance counter and averages this over five samples:

<Rule ID="Microsoft.Samples.ProcessCollection" Target="Microsoft.Samples.ApplicationX">

<Category>Custom</Category>

<DataSources>

<DataSource ID="PerfDS" TypeID="Perf!System.Performance.OptimizedDataProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<CounterName><![CDATA[% Processor Time]]></CounterName>

<ObjectName>Process</ObjectName>

<InstanceName>ApplicationX</InstanceName>

<AllInstances>false</AllInstances>

<Frequency>60</Frequency>

<Tolerance>0</Tolerance>

<ToleranceType>Absolute</ToleranceType>

<MaximumSampleSeparation>1</MaximumSampleSeparation>

</DataSource>

</DataSources>

<ConditionDetection ID="Averager" TypeID="Perf!System.Performance.AveragerCondition">

<NumSamples>5</NumSamples>

</ConditionDetection>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectPerformanceData" />

</WriteActions>

</Rule>

Any data source that generates performance data can be used with the averager module. The following example shows a script-based data source that maps data to a performance counter before using the averager module. The rule is configured to average over two samples, and it writes the data to the Operations Manager database only:

<Rule ID="Microsoft.Samples.PerfMapperSingleAverager" Target="Microsoft.Samples.ApplicationX">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Script" TypeID="Windows!Microsoft.Windows.TimedScript.PerformanceProvider">

<IntervalSeconds>30</IntervalSeconds>

<SyncTime />

<ScriptName>Microsoft.Samples.PerfMapper.vbs</ScriptName>

<Arguments />

<ScriptBody>

Option Explicit

Dim OMApi, OMPBag

Set OMApi = CreateObject("MOM.ScriptAPI")

Set OMPBag = OMApi.CreatePropertyBag()

Randomize()

Call OMPBag.AddValue("Value", CInt(Rnd()\*100))

Call OMAPI.Return(OMPBag)

</ScriptBody>

<TimeoutSeconds>10</TimeoutSeconds>

<ObjectName>AppplicationX</ObjectName>

<CounterName>Averager Test</CounterName>

<InstanceName />

<Value>$Data/Property[@Name='Value']$</Value>

</DataSource>

</DataSources>

<ConditionDetection ID="Averager" TypeID="Perf!System.Performance.AveragerCondition">

<NumSamples>2</NumSamples>

</ConditionDetection>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectPerformanceData" />

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.Performance.DataGenericMapper

The System.Performance.DataGenericMapper module is a condition detection module type that is used to map data of any type to performance data that can be stored in the Operations Manager database and the data warehouse. Only event data, performance data, state change data, and alert data can be stored natively in the databases, so it might be necessary to map data to a supported type.

Parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ObjectName | String | Contains the performance object name. |
| CounterName | String | Contains the performance counter name. |
| InstanceName | String | Contains the instance name. This element is required, but can be empty if there is only a single instance. |
| Value | String | Contains the numeric value for the performance value. This is defined as a String in the module schema, but it should be a numeric value. |

All parameters are required, and no parameters are defined as overrideable.

Remarks

It is necessary to map data to performance data only if you want to store the data in the database as a performance value. In monitoring scenarios that require you to use the data only to trigger a state change on a monitor or generate an alert, you do not have to store the data if you do not want to. The raw data item is stored as the context of a state change or alert, and it appears in the Operations console. It is necessary to map the data and store only if you want to report on or view the data on an ongoing basis.

You can see the performance counters generated by the mapper module and written to the database in the same way a standard Windows performance counter is seen by using the Operations console or performance reports.

Any data reference that is used in the mapper configuration is dependent on the data type that is the input to the mapper module.

Examples

The following example shows a rule that executes a script on a timed interval. The data in the property bag output from the script is used to generate a single performance point and store it in the Operations Manager database. The script generates a random value between 0 and 100. Generally for this scenario, you should use the Microsoft.Windows.TimedScript.PerformanceProvider module type that is defined in the Microsoft.Windows.Library because this provides the composition of the script and the mapper in a data source. This following example is only for illustration purposes.

<Rule ID="Microsoft.Samples.PerfMapperSingle" Target="Microsoft.Samples.ApplicationX" Enabled="false">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Script" TypeID="Windows!Microsoft.Windows.TimedScript.PropertyBagProvider">

<IntervalSeconds>60</IntervalSeconds>

<SyncTime/>

<ScriptName>Microsoft.Samples.PerfMapper.vbs</ScriptName>

<Arguments/>

<ScriptBody>

<![CDATA[

Option Explicit

Dim OMApi, OMPBag

Set OMApi = CreateObject("MOM.ScriptAPI")

Set OMPBag = OMApi.CreatePropertyBag()

Call OMPBag.AddValue("CounterName", "Test Counter 1")

Randomize()

Call OMPBag.AddValue("Value", CInt(Rnd()\*100))

Call OMAPI.Return(OMPBag)

]]>

</ScriptBody>

<TimeoutSeconds>30</TimeoutSeconds>

</DataSource>

</DataSources>

<ConditionDetection ID="Mapper" TypeID="Perf!System.Performance.DataGenericMapper">

<ObjectName>AppplicationX</ObjectName>

<CounterName>$Data/Property[@Name='CounterName']$</CounterName>

<InstanceName/>

<Value>$Data/Property[@Name='Value']$</Value>

</ConditionDetection>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectPerformanceData"/>

</WriteActions>

</Rule>

The rule uses data from the script to set the counter name and the value of the performance counter. The following code example is an example data item from the script that is used by the mapper:

<DataItem type="System.PropertyBagData" time="2008-10-17T09:50:34.8054044-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<Property Name="CounterName" VariantType="8">Test Counter 1</Property>

<Property Name="Value" VariantType="2">29</Property>

</DataItem

The following code example shows that this data item is mapped to the following performance data item and stored in the database:

<DataItem type="System.Performance.Data" time="2008-10-17T09:55:50.0519259-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818">

<TimeSampled>2008-10-17T09:55:50.0429250-07:00</TimeSampled>

<ObjectName>AppplicationX</ObjectName>

<CounterName>Test Counter 1</CounterName>

<InstanceName></InstanceName>

<IsNull Type="Boolean">false</IsNull>

<Value>13</Value>

</DataItem>

A data source that outputs multiple data items can be used with this module type to create multiple performance values during a single execution. The following code example shows a rule that executes a script on a timed interval. The script outputs four property bags; each property bag is mapped to an instance of a performance counter.

<Rule ID="Microsoft.Samples.PerfMapperMulti" Target="Microsoft.Samples.ApplicationX" Enabled="false">

<Category>Custom</Category>

<DataSources>

<DataSource ID="Script" TypeID="Windows!Microsoft.Windows.TimedScript.PropertyBagProvider">

<IntervalSeconds>60</IntervalSeconds>

<SyncTime/>

<ScriptName>Microsoft.Samples.PerfMapper.vbs</ScriptName>

<Arguments/>

<ScriptBody>

<![CDATA[

Option Explicit

Dim OMApi, OMPBag

Set OMApi = CreateObject("MOM.ScriptAPI")

Randomize()

Dim i

For i = 1 to 4

Set OMPBag = OMApi.CreatePropertyBag()

Call OMPBag.AddValue("CounterName", "Test Counter 2")

Call OMPBag.AddValue("InstanceName", "Instance" & i)

Call OMPBag.AddValue("Value", CInt(Rnd()\*100))

Call OMApi.AddItem(OMPBag)

Next

Call OMAPI.ReturnItems()

]]>

</ScriptBody>

<TimeoutSeconds>30</TimeoutSeconds>

</DataSource>

</DataSources>

<ConditionDetection ID="Mapper" TypeID="Perf!System.Performance.DataGenericMapper">

<ObjectName>AppplicationX</ObjectName>

<CounterName>$Data/Property[@Name='CounterName']$</CounterName>

<InstanceName>$Data/Property[@Name='InstanceName']$</InstanceName>

<Value>$Data/Property[@Name='Value']$</Value>

</ConditionDetection>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectPerformanceData"/>

</WriteActions>

</Rule>

Related Modules

There are a number of other module types that use this module type as part of a composition that might be more suitable to your scenario. Use these specialized module types where it is possible, rather than declaring your own compositions.

|  |  |  |
| --- | --- | --- |
| Module | Defined In | Description |
| Microsoft.Windows.TimedScript.PerformanceProvider | Windows.Library | Generated performance data from a timed script. |
| Microsoft.Windows.WmiPerfCounterProvider | Windows.Library | Generated performance data from a timed Windows Management Instrumentation (WMI) query. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.Performance.OptimizedDataProvider

The System.Performance.OptimizedDataProvider is a data source module type that outputs Windows performance data on a timed interval.

Parameters

The System.Performance.OptimizedDataProvider module takes the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ComputerName | String | Contains the name of the computer to collect performance data from. |
| CounterName | String | Specifies the name of the Windows performance counter. |
| ObjectName | String | Specifies the Windows performance object name. |
| InstanceName | String | Contains the Windows performance instance, if applicable. |
| AllInstances | Boolean | Specifies whether to collect all instances for a multi-instance counter. |
| Frequency | Unsigned Integer | Specifies the interval in seconds between performance data collection. Data might not actually be output from the module on this frequency if optimization is being used. |
| Tolerance | Double | Specifies the delta change in value that causes a collection to happen. |
| ToleranceType | ToleranceType | Specifies the type of value that is specified in the tolerance parameter. This value can be either Absolute or Percentage. |
| MaximumSampleSeparation | Unsigned Integer | Specifies the maximum separation between samples when using optimization. |

ComputerName

The ComputerName configuration is most commonly passed to the module that is using the $Target notation, although it can be specified as any string. This module attempts to connect to either the local computer’s event log or a remove event log, depending on the configuration that is passed.

For a workflow that is targeted at a Windows-based computer, the ComputerName element is specified as:

<ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

If the target is a class that hosted by a computer, hosting references are added as required. For example:

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

Counter Selection

The CounterName and ObjectName parameters identify the Windows performance counter to collect. The values entered here depend on the performance counters that are running on the computer that is being targeted with the workflow that contains the module. The Windows performance monitor can be used to identify available performance counters.

For example, the following configuration collects the average disk queue length on the logical disk object:

<CounterName>Avg. Disk Queue Length</CounterName>

<ObjectName>LogicalDisk</ObjectName>

The InstanceName and AllInstances parameters specify how to handle counters that have multiple instances. For example, the logical disk counter has an instance in Windows performance monitor for each available logical disk.

You can specify a single instance by using the InstanceName parameter, or you can specify AllInstances. To specify a single instance:

<CounterName>Avg. Disk Queue Length</CounterName>

<ObjectName>LogicalDisk</ObjectName>

<InstanceName>C:</InstanceName>

<AllInstances>false</AllInstances>

To specify all instances, leave the InstanceName parameter empty and set the AllInstances parameter to true. For example:

<CounterName>Avg. Disk Queue Length</CounterName>

<ObjectName>LogicalDisk</ObjectName>

<InstanceName/>

<AllInstances>true</AllInstances>

For a counter that does not have any instances, leave the InstanceName parameter empty, and set the AllInstances parameter to false. For example:

<CounterName>Available MBytes</CounterName>

<ObjectName>Memory</ObjectName>

<InstanceName />

<AllInstances>false</AllInstances>

For multi-instance counters, you might want to collect the performance data against specific object instances in Operations Manager. In this situation, it is common to use a $Target expression for the instance name to collect only the relevant instance for the given discovered object. For example:

<CounterName>Avg. Disk sec/Write</CounterName>

<ObjectName>LogicalDisk</ObjectName>

<InstanceName>$Target/Property[Type="Windows!Microsoft.Windows.LogicalDevice"]/DeviceID$</InstanceName>

<AllInstances>false</AllInstances>

Optimization

The System.Performance.OptimizedDataProvide module type can output both optimized and unoptimized data. Unoptimized data results in a performance data item being returned once every specified polling interval. Use this when you want to collect every performance data value, regardless of whether the value has changed since the last polling interval.

To specify unoptimized collection, set the Tolerance parameter to 0 and set the MaximumSampleSeparation to 1. For example:

<ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<CounterName>Available MBytes</CounterName>

<ObjectName>Memory</ObjectName>

<InstanceName />

<AllInstances>false</AllInstances>

<Frequency>120</Frequency>

<Tolerance>0</Tolerance>

<ToleranceType>Absolute</ToleranceType>

<MaximumSampleSeparation>1</MaximumSampleSeparation>

The recommended approach to performance collection is to use optimization to reduce the amount of data that is stored in the Operations Manager database and data warehouse. You do this by specifying a threshold (the tolerance) that causes a new data point to be collected. For example, if an absolute tolerance of 10 is specified, a new data item is not generated unless the new performance data differs in value by at least -10 or +10. The tolerance can be specified in absolute value or in a percentage change.

The MaximumSampleSeparation parameter is used to ensure that data is collected at least once in a given period of time. If a performance counter does not change for 24 hours, the module does not output a new data point, which might lead to issues in reporting. Specifying a value in this parameter forces the module to output a data point after a set number of samples, even if the tolerance has not met the criteria. The separation is expressed as a multiple of the frequency. For example, if the frequency is set to 120 seconds, and the MaximumSampleSeparation parameter is set to 20, a data item is output at least every 40 minutes.

The following configuration example shows an optimized configuration:

<ComputerName>$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<CounterName>Available MBytes</CounterName>

<ObjectName>Memory</ObjectName>

<InstanceName />

<AllInstances>false</AllInstances>

<Frequency>120</Frequency>

<Tolerance>10</Tolerance>

<ToleranceType>Percentage</ToleranceType>

<MaximumSampleSeparation>20</MaximumSampleSeparation>

Remarks

If a workflow that uses this module type executes on a computer that does not have the counter, warning events are logged in the Operations Manager event log indicating that the performance counter cannot be found. You should use targeting to ensure that your workflows run only on computers that are known to have the counters present. For example, when you collect SQL Server performance data, you should target specific SQL Server classes such as, Microsoft.SQLServer.2005.DBEngine rather than Microsoft.Windows.Computer

It is valid to use $Target properties in the CounterName and ObjectName parameters in addition to the InstanceName parameter. This is required if the object or counter name in Windows performance monitor is variable. An example of this is Microsoft SQL Server. The object names have the instance name as part of the name, so it is required to use a discovered property for configuration.

For example:

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<CounterName>Logins/sec</CounterName>

<ObjectName>$Target/Property[Type="SQL!Microsoft.SQLServer.DBEngine"]/PerformanceCounterObject$:General Statistics</ObjectName>

<InstanceName />

<AllInstances>false</AllInstances>

<Frequency>120</Frequency>

Examples

The following code example shows an un-optimized performance collection rule. This rule collects a counter with a single instance on a 15-minute interval, and it writes the data to both the Operations Manager database and the data warehouse.

<Rule ID="Microsoft.Samples.CollectThreads" Target="Microsoft.Samples.ApplicationX">

<Category>Custom</Category>

<DataSources>

<DataSource ID="DS" TypeID="Perf!System.Performance.OptimizedDataProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<CounterName>Threads</CounterName>

<ObjectName>System</ObjectName>

<InstanceName />

<AllInstances>false</AllInstances>

<Frequency>900</Frequency>

<Tolerance>0</Tolerance>

<ToleranceType>Absolute</ToleranceType>

<MaximumSampleSeparation>1</MaximumSampleSeparation>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectPerformanceData" />

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishPerformanceData" />

</WriteActions>

</Rule>

The following example shows an optimized performance collection rule that collects a counter with multiple instances and optimizes the collection by using an absolute tolerance. The rule uses $Target properties to associate the collected values with a specific instance.

<Rule ID="Microsoft.Windows.Server.2003.LogicalDisk.FreeMB.Collection" Target="Microsoft.Windows.Server.2003.LogicalDisk">

<Category>PerformanceCollection</Category>

<DataSources>

<DataSource ID="PerformanceDS" TypeID="SystemPerf!System.Performance.OptimizedDataProvider">

<ComputerName>$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$</ComputerName>

<CounterName>Free Megabytes</CounterName>

<ObjectName>LogicalDisk</ObjectName>

<InstanceName>$Target/Property[Type="Windows!Microsoft.Windows.LogicalDevice"]/DeviceID$</InstanceName>

<AllInstances>false</AllInstances>

<Frequency>300</Frequency>

<Tolerance>100</Tolerance>

<ToleranceType>Absolute</ToleranceType>

<MaximumSampleSeparation>12</MaximumSampleSeparation>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectPerformanceData" />

<WriteAction ID="WriteToDW" TypeID="SCDW!Microsoft.SystemCenter.DataWarehouse.PublishPerformanceData" />

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.Scheduler

This data source module is used to schedule workflows and outputs a System.Trigger type. It is defined in System.Library.

Parameters

The System.Scheduler data source module takes the following parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Scheduler | PublicSchedulerType | The recurring schedule to output data items |

The System.Scheduler module takes a single parameter called Scheduler that is of complex type PublicSchedulerType. There are a number of scheduling scenarios that require different configurations of this Scheduler parameter.

Simple Recurring Schedule

A simple schedule specifies a unit of time (seconds, by default) and a recurring interval to execute on. The scheduler outputs a data item the first time it is initialized (the first time the workflow loads, after the health service restarts, or after exiting maintenance mode) and then outputs data items at the specified frequency.

The following code example is a configuration for a simple recurring schedule that executes every 30 seconds:

<Scheduler>

<SimpleRecurringSchedule>

<Interval Unit="Seconds">30</Interval>

<SyncTime/>

</SimpleReccuringSchedule>

<ExcludeDates/>

</Scheduler>

The Unit attribute is optional, and if it is excluded, the module uses seconds by default. For example:

<Scheduler>

<SimpleRecurringSchedule>

<Interval>30</Interval>

<SyncTime/>

</SimpleReccuringSchedule>

<ExcludeDates/>

</Scheduler>

The allowed values for the unit attributes are:

 Seconds

 Minutes

 Hours

 Days

Simple Synchronized Recurring Schedule

The simple recurring schedule can be extended by adding a synchronization time. Specifying a synchronization time forces the module to output a data item at the specified time, and it executes on that frequency based on that synchronization point.

The following code example shows the module configured to run every 15 minutes and synchronize at 13:00. Note that the synchronization time is specified by using a 24-hour format.

<Scheduler>

<SimpleRecurringSchedule>

<Interval Unit="Minutes">15</Interval>

<SyncTime>13:00</SyncTime>

</SimpleReccuringSchedule>

<ExcludeDates/>

</Scheduler>

The module does not have to wait until the synchronization time for the first output. When it is initialized, the full schedule is calculated, and the module is output at the next closest time. For example, if the preceding configuration was used and the workflow was initialized at 12:16, the first data item would be output at 12:30 because this is on the required schedule.

Weekly Schedule

A weekly schedule can be used to trigger the data item output at specific times during the day. Multiple sets of trigger points can be added as required.

The following code example shows the module configured to execute at 1:00 on Monday:

<Scheduler>

<WeeklySchedule>

<Windows>

<Daily>

<Start>01:00</Start>

<End>01:00</End>

<DaysOfWeekMask>2</DaysOfWeekMask>

</Daily>

</Windows>

</WeeklySchedule>

<ExcludeDates/>

</Scheduler>

Note

the PublicScheduleType definition is shared between multiple module types. To use them in the System.Scheduler type, set the Start and End time elements to be the same.

The DaysOfWeekMask element represents the days that the module should trigger output. A single day or multiple days can be specified according to the following values:

|  |  |
| --- | --- |
| Day | Value |
| Sunday | 1 |
| Monday | 2 |
| Tuesday | 4 |
| Wednesday | 8 |
| Thursday | 16 |
| Friday | 32 |
| Saturday | 64 |

To specify a single day, enter the enumerator value for that day directly into the DaysOfWeekMask configuration element.

To specify multiple days, add the enumerator values for the days together. For example, for Monday, Wednesday, and Friday, specify 42 (2+8+32).

You can specify multiple days and times to execute on by using the weekly schedule. Any number of daily elements can be added to the configuration.

The following code example shows a schedule that is configured to run once a day at 1:00 on Monday, Wednesday, and Friday, and at 3:00 on Tuesday and Thursday:

<Scheduler>

<WeeklySchedule>

<Windows>

<Daily>

<Start>01:00</Start>

<End>01:00</End>

<DaysOfWeekMask>42</DaysOfWeekMask>

</Daily>

<Daily>

<Start>03:00</Start>

<End>03:00</End>

<DaysOfWeekMask>20</DaysOfWeekMask>

</Daily>

</Windows>

</WeeklySchedule>

<ExcludeDates/>

</Scheduler>

Note

The PublicSchedulerType exposes a MultipleDays element. Do not use this element for System.Scheduler modules.

Excluding Specific Dates

You can exclude specific dates when you are using either the simple recurring schedule or the weekly schedule. Specify individual days by using the ExcludeDates configuration. You can add any number of date ranges, but a date range must be contiguous (for example, February 2 to February 4).

Do not specify the year when you enter the date. You can specify a single date by setting the start and end date to be the same date. Optionally, you can use the Description to describe why the exclusion has been made. The supplied description is shown in the configuration user interface for the module.

The following code example shows a simple schedule with a single day (January 12th) excluded:

<Scheduler>

<SimpleReccuringSchedule>

<Interval Unit="Minutes">15</Interval>

</SimpleReccuringSchedule>

<ExcludeDates>

<DayInterval>

<Start>01/12</Start>

<End>01/12</End>

<Description/>

</DayInterval>

</ExcludeDates>

</Scheduler>

The following code example is more complex, with multiple date ranges excluded and a weekly schedule set:

<Scheduler>

<WeeklySchedule>

<Windows>

<Daily>

<Start>14:00</Start>

<End>14:00</End>

<DaysOfWeekMask>42</DaysOfWeekMask>

</Daily>

</Windows>

</WeeklySchedule>

<ExcludeDates>

<DayInterval>

<Start>02/02</Start>

<End>02/05</End>

<Description />

</DayInterval>

<DayInterval>

<Start>12/01</Start>

<End>12/01</End>

<Description />

</DayInterval>

</ExcludeDates>

</Scheduler>

Remarks

The System.Scheduler is used when you want to start a workflow on a scheduled, repeating interval. This module is used as a base for all timed discoveries, rules and monitors. After initialization and on the specified interval, the data source outputs a System.Trigger data item that is passed to the next module in the workflow. The subsequent module in the workflow must be defined as taking either a System.BaseData or System.TriggerData type.

The data item that the module outputs is most commonly used only to trigger the execution of a subsequent module, and the actual data that is contained in the trigger data item is not used. The only data that is contained in the trigger data item is the time of the trigger and the health service ID that generated the data item. The following code example shows a System.TriggerData data item:

<DataItem type="System.TriggerData" time="2008-10-06T12:24:33.0056793-07:00" sourceHealthServiceId="B0BE86FA-56AD-1F2E-EE87-8DF72FC53818"/>

Examples

The following code example shows a rule that runs on a 5-minute schedule and executes a script. The script logs an event to the Operations Manager event log on when the script completes.

<Rule ID="Microsoft.Samples.ExecuteTestScript" Enabled="true" Target="Microsoft.Samples.ApplicationX" >

<Category>Custom</Category>

<DataSources>

<DataSource ID="Scheduler" TypeID="System!System.Scheduler">

<Scheduler>

<SimpleReccuringSchedule>

<Interval Unit="Minutes">5</Interval>

</SimpleReccuringSchedule>

<ExcludeDates/>

</Scheduler>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="Script" TypeID="Windows!Microsoft.Windows.ScriptWriteAction">

<ScriptName>Microsoft.Samples.LogOpsMgrEvent.vbs</ScriptName>

<Arguments/>

<ScriptBody>

Option Explicit

Dim OMAPI

Set OMAPI = CreateObject("MOM.ScriptAPI")

Call OMAPI.LogScriptEvent("Microsoft.Samples.LogOpsMgrEvent",101,0,"Script Executed")

</ScriptBody>

<TimeoutSeconds>30</TimeoutSeconds>

</WriteAction>

</WriteActions>

</Rule>

Related Modules

The System.Scheduler has no overrideable parameters due to the complexity of the configuration, so any workflows that use this module directly cannot have their schedule information overridden. For this reason, a number of simpler modules have been defined for common scenarios. The following table summarizes these modules:

|  |  |  |
| --- | --- | --- |
| Module Type | Define In | Usage |
| System.SimpleScheduler | System.Library | Simple recurring schedule where only an interval and a synchronization time is required |
| System.Discovery.Scheduler | System.Library | Discovery workflows only. Using this module type will mean the workflow is never cooked down with other scheduler-based workflows. This provides a simple recurring schedule. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpProbe

The System.SnmpProbe probe action module type is used to perform SNMP GET, GET-NEXT and SET actions and returns an System.SnmpData data object.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Walk | Boolean | Optional. Specifies whether to walk or perform a GET-NEXT action on the SNMP MIB.The default value is false. |
| WalkReturnMultipleItems | Boolean | R2 Specific. Optional. Specifies whether to return separate System.SnmpData objects for each traversed SNMP MIB OID. The default value is false. |
| IsWriteAction | Boolean | Specifies whether the current operation on the SNMP MIB is a write action. When set to true, the System.SnmpProbe module is configured to perform a SET operation on the SNMP MIB. When false, the module is intended to perform either a GET or GET-NEXT operation. |
| IP | String | Specifies the IP address of the SNMP managed entity. |
| CommunityString | String | Contains the SNMP Community.. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| SnmpVarBinds | Complex | Contains a list of SNMP variable bindings as SnmpVarBind elements. If the Walk parameter is true, only one SnmpVarBind must be specified. |

Walk

When set to true, the System.SnmpProbe module is configured to walk the SNMP MIB in a GET-NEXT request. The SnmpVarBinds list must contain only one SnmpVarBind element from which to begin the walk. When false, the module is intended to perform a GET or SET operation on specified SNMP MIB variables.

<Walk>true</Walk>

WalkReturnMultipleItems

When set to true, the probe module will return each traversed OID instance in a separate System.SnmpData object. This allows the management server to create separate entities for each discovered instance. When false, the probe module returns a single System.SnmpData object. This parameter is used only when the Walk parameter is set to true.

Note

This parameter exists only for the R2 release of Operations Manager.

<WalkReturnMultipleItems>true</WalkReturnMultipleItems>

IP

This is the IP address of the SNMP MIB..

<IP>$Target/Property[Type="Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

SnmpVarBinds

The SnmpVarBinds element contains an SMTP variable binding list. Each variable binding in the list is represented as an SnmpVarBind element. The SnmpVarBind element contains the following sub elements:

|  |  |  |
| --- | --- | --- |
| Element | Type | Description |
| OID | String | Specifies the SNMP OID of the entity in the SNMP MIB table. |
| Value | String | The value and variable type of OID entity. The variable type can be specified in the optional VariantType property. The value of the Value element is set only when the IsWriteAction parameter is set to true and a SET action is to be performed.  The following table maps SNMP data types to VariantType strings:   |  |  |  | | --- | --- | --- | | SNMP MIB type | VariantType Value | Variant Type | | Integer, Counters, Gauges, TimeTicks (SNMPv1 and SNMPv2) | 3 | VT\_I4 | | Octet strings, Object Identifiers, IP Addresses, Opaques, Network Addresses | 8 | VT\_BSTR | | Null | 1 | VT\_NULL | |
| Syntax | String | The Syntax property indicates the failure or success of the SNMP operation and can be inspected by a condition detection module in a workflow. If the operation is successful, the module sets the value of this property to the appropriate SNMP syntax data type.  Note  Note that for SET operations, the Syntax property must be set to one of the SNMP syntax data types.  The following table maps SNMP syntax data types to the appropriate string values.   |  |  | | --- | --- | | Syntax data type | String value | | SNMP\_SYNTAX\_INT | 2 | | SNMP\_SYNTAX\_INT32 | 2 | | SNMP\_SYNTAX\_BITS | 3 | | SNMP\_SYNTAX\_OCTETS | 4 | | SNMP\_SYNTAX\_NULL | 5 | | SNMP\_SYNTAX\_OID | 6 | | SNMP\_SYNTAX\_NSAPADDR | 8 | | SNMP\_SYNTAX\_SEQUENCE | 48 | | SNMP\_SYNTAX\_IPADDR | 64 | | SNMP\_SYNTAX\_CNTR32 | 65 | | SNMP\_SYNTAX\_GUAGE32 | 66 | | SNMP\_SYNTAX\_TIMETICKS | 67 | | SNMP\_SYNTAX\_OPAQUE | 68 | | SNMP\_SYNTAX\_NSAPADDR | 69 | | SNMP\_SYNTAX\_CENTR64 | 70 | | SNMP\_SYNTAX\_UINT32 | 71 |     The following table maps SNMP error syntax data types to the appropriate string values.   |  |  | | --- | --- | | Error syntax data type | String value | | SNMP\_SYNTAX\_NOSUCHOBJECT | 128 | | SNMP\_SYNTAX\_NOSUCHINSTANCE | 129 | | SNMP\_SYNTAX\_ENDOFMIBVIEW | 130 |     For more information, see RFC 1902, "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)." |

The following examples demonstrate the appropriate syntax for defining SnmpVarBind elements for both SET and GET actions.

In the following sample code, an SnmpVarBind element is defined for a probe that is to perform a GET action on the sysDescr MIB element.

<SnmpVarBind>

<OID>1.3.6.1.2.1.1.0</OID>

<Syntax>0</Syntax>

<Value VariantType=”8”/>

</SnmpVarBind>

In the following sample code, an SnmpVarBind element is defined for a probe that is to perform a SET action on the sysName MIB element.

<SnmpVarBind>

<OID>1.3.6.1.21.1.5.0</OID>

<Syntax>4</Syntax>

<Value VariantType=”8”>New System Name</Value>

</SnmpVarBind>

Remarks

The System.SnmpProbe module wraps portions of the WinSNMP API. For a deeper understanding of how the module works internally, see the [WinSNMP API topic](http://go.microsoft.com/fwlink/?LinkID=151855&clcid=0x409).

Examples

The following XML code example shows a successful data item output from the System.SnmpProbe module.

<?xml version="1.0" encoding="utf-8"?>

<DataItem type="System.SnmpData">

<SnmpData>

<Source>127.0.0.1</Source>

<Destination>127.0.0.9</Destination>

<CommunityString>public</CommunityString>

<ErrorCode>1</ErrorCode>

<Version>1</Version>

<SnmpVarBinds>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.1.5.0</OID>

<Syntax>4</Syntax>

<Value VariantType="8" >My Machine Name</Value>

</SnmpVarBind>

</SnmpVarBinds>

</SnmpData>

</DataItem>

The following sample management pack illustrates how you can use the System.SnmpProbe to discover a specific kind of SNMP device, namely, a Microsoft.OM.Samples.Modules.SNMP.WindowsServerDevice. The example also shows how you can use the System.SnmpProbe module in a task that targets the discovered SNMP device and performs a GET on the SysObjectID field.

<ManagementPack ContentReadable="true" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<Manifest>

<Identity>

<ID>Microsoft.OM.Samples.Modules.SNMP</ID>

<Version>1.0.0.0</Version>

</Identity>

<Name>Microsoft.OM.Samples.Modules.SNMP</Name>

<References>

<Reference Alias="SC">

<ID>Microsoft.SystemCenter.Library</ID>

<Version>6.1.7043.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="MicrosoftSystemCenterNetworkDeviceLibrary">

<ID>Microsoft.SystemCenter.NetworkDevice.Library</ID>

<Version>6.1.7043.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="Windows">

<ID>Microsoft.Windows.Library</ID>

<Version>6.1.7043.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="Health">

<ID>System.Health.Library</ID>

<Version>6.1.7043.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="System">

<ID>System.Library</ID>

<Version>6.1.7043.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

<Reference Alias="Snmp">

<ID>System.Snmp.Library</ID>

<Version>6.1.7043.0</Version>

<PublicKeyToken>31bf3856ad364e35</PublicKeyToken>

</Reference>

</References>

</Manifest>

<TypeDefinitions>

<EntityTypes>

<ClassTypes>

<ClassType ID="Microsoft.OM.Samples.Modules.SNMP.WindowsServerDevice" Accessibility="Internal" Abstract="false" Base="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Hosted="false" Singleton="false" />

</ClassTypes>

</EntityTypes>

<ModuleTypes>

<DataSourceModuleType ID="Microsoft.OM.Samples.Modules.SNMP.SNMPDeviceDiscoveryDataSource" Accessibility="Internal" Batching="false">

<Configuration>

<IncludeSchemaTypes>

<SchemaType>System!System.Discovery.MapperSchema</SchemaType>

<SchemaType>System!System.ExpressionEvaluatorSchema</SchemaType>

<SchemaType>System!System.ParamListSchema</SchemaType>

</IncludeSchemaTypes>

<xsd:element minOccurs="1" name="Interval" type="xsd:integer" />

<xsd:element minOccurs="1" name="IP" type="xsd:string" />

<xsd:element minOccurs="1" name="CommunityString" type="xsd:string" />

<xsd:element minOccurs="1" name="ClassId" type="xsd:string" />

<xsd:element minOccurs="1" name="Expression" type="ExpressionType" />

<xsd:element minOccurs="1" name="InstanceSettings" type="SettingsType" />

<xsd:element name="SnmpVarBinds">

<xsd:complexType>

<xsd:sequence>

<xsd:element minOccurs="1" maxOccurs="unbounded" name="SnmpVarBind">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="OID" type="xsd:string" />

<xsd:element name="Syntax" type="xsd:integer" />

<xsd:element name="Value">

<xsd:complexType>

<xsd:simpleContent>

<xsd:extension base="xsd:string">

<xsd:attribute name="VariantType" type="xsd:integer" use="optional" />

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</Configuration>

<ModuleImplementation Isolation="Any">

<Composite>

<MemberModules>

<DataSource ID="Scheduler" TypeID="System!System.Scheduler">

<Scheduler>

<SimpleReccuringSchedule>

<Interval Unit="Seconds">$Config/Interval$</Interval>

</SimpleReccuringSchedule>

<ExcludeDates />

</Scheduler>

</DataSource>

<ProbeAction ID="Probe" TypeID="Snmp!System.SnmpProbe">

<IsWriteAction>false</IsWriteAction>

<IP>$Config/IP$</IP>

<CommunityString>$Config/CommunityString$</CommunityString>

<SnmpVarBinds>$Config/SnmpVarBinds$</SnmpVarBinds>

</ProbeAction>

<ConditionDetection ID="Filter" TypeID="System!System.ExpressionFilter">

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">ErrorCode</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<XPathQuery Type="UnsignedInteger">1</XPathQuery>

</ValueExpression>

</SimpleExpression>

</Expression>

</ConditionDetection>

<ConditionDetection ID="Mapper" TypeID="System!System.Discovery.FilteredClassSnapshotDataMapper">

<Expression>$Config/Expression$</Expression>

<ClassId>$Config/ClassId$</ClassId>

<InstanceSettings>$Config/InstanceSettings$</InstanceSettings>

</ConditionDetection>

</MemberModules>

<Composition>

<Node ID="Mapper">

<Node ID="Filter">

<Node ID="Probe">

<Node ID="Scheduler" />

</Node>

</Node>

</Node>

</Composition>

</Composite>

</ModuleImplementation>

<OutputType>System!System.Discovery.Data</OutputType>

</DataSourceModuleType>

</ModuleTypes>

</TypeDefinitions>

<Monitoring>

<Discoveries>

<Discovery ID="Microsoft.OM.Samples.Modules.SNMP.DiscoverWindowsServer" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" ConfirmDelivery="true" Remotable="true" Priority="Normal">

<Category>Discovery</Category>

<DiscoveryTypes>

<DiscoveryClass TypeID="Microsoft.OM.Samples.Modules.SNMP.WindowsServerDevice" />

</DiscoveryTypes>

<DataSource ID="DiscoveryDS" TypeID="Microsoft.OM.Samples.Modules.SNMP.SNMPDeviceDiscoveryDataSource">

<Interval>300</Interval>

<IP>$Target/Property[Type="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>public</CommunityString>

<ClassId>$MPElement[Name="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]$</ClassId>

<Expression>

<RegExExpression>

<ValueExpression>

<XPathQuery>/SnmpVarBinds/SnmpVarBind[1]/Value</XPathQuery>

</ValueExpression>

<Operator>ContainsSubstring</Operator>

<Pattern>1.3.6.1.4.1.311.1.1.3.1.2</Pattern>

</RegExExpression>

</Expression>

<InstanceSettings>

<Settings>

<Setting>

<Name>$MPElement[Name="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</Name>

<Value>$Data/Source$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/Name$</Name>

<Value>$Data/SnmpVarBinds/SnmpVarBind[5]/Value$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/SystemDescription$</Name>

<Value>$Data/SnmpVarBinds/SnmpVarBind[4]/Value$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/SystemContact$</Name>

<Value>$Data/SnmpVarBinds/SnmpVarBind[3]/Value$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/SystemLocation$</Name>

<Value>$Data/SnmpVarBinds/SnmpVarBind[2]/Value$</Value>

</Setting>

<Setting>

<Name>$MPElement[Name="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/SystemOID$</Name>

<Value>$Data/SnmpVarBinds/SnmpVarBind[1]/Value$</Value>

</Setting>

</Settings>

</InstanceSettings>

<SnmpVarBinds>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.1.5.0</OID>

<Syntax>0</Syntax>

<Value VariantType="8" />

</SnmpVarBind>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.1.1.0</OID>

<Syntax>0</Syntax>

<Value VariantType="8" />

</SnmpVarBind>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.1.4.0</OID>

<Syntax>0</Syntax>

<Value VariantType="8" />

</SnmpVarBind>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.1.6.0</OID>

<Syntax>0</Syntax>

<Value VariantType="8" />

</SnmpVarBind>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.1.2.0</OID>

<Syntax>0</Syntax>

<Value VariantType="8" />

</SnmpVarBind>

</SnmpVarBinds>

</DataSource>

</Discovery>

</Discoveries>

<Tasks>

<Task ID="Microsoft.Samples.Snmp.SnmpProbeTask" Accessibility="Internal" Enabled="true" Target="Microsoft.OM.Samples.Modules.SNMP.WindowsServerDevice" Timeout="300" Remotable="true">

<Category>Custom</Category>

<ProbeAction ID="Probe" TypeID="Snmp!System.SnmpProbe">

<IsWriteAction>false</IsWriteAction>

<IP>$Target/Property[Type="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>public</CommunityString>

<SnmpVarBinds>

<SnmpVarBind>

<OID>1.3.6.1.4.1.311.1.1.3.1.2</OID>

<Syntax>0</Syntax>

<Value VariantType="8" />

</SnmpVarBind>

</SnmpVarBinds>

</ProbeAction>

</Task>

</Tasks>

</Monitoring>

<LanguagePacks>

<LanguagePack ID="ENU" IsDefault="true">

<DisplayStrings>

<DisplayString ElementID="Microsoft.OM.Samples.Modules.SNMP">

<Name>Microsoft OM SNMP Module Samples</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.OM.Samples.Modules.SNMP.DiscoverWindowsServer">

<Name>Discover Windows Server</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.OM.Samples.Modules.SNMP.SNMPDeviceDiscoveryDataSource">

<Name>Windows Server SNMP Discovery DataSource</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.OM.Samples.Modules.SNMP.WindowsServerDevice">

<Name>Windows Server Device</Name>

</DisplayString>

<DisplayString ElementID="Microsoft.Samples.Snmp.SnmpProbeTask">

<Name>SnmpProbe Task</Name>

<Description />

</DisplayString>

</DisplayStrings>

</LanguagePack>

</LanguagePacks>

</ManagementPack>

Related Modules

There are a number of other module types that use this module type as part of a composition that might be more suitable to your scenario. You should use these specialized module types where it is possible rather than declaring your own compositions:

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| System.SnmpEventProvider | System.Snmp.Library | Outputs System.Event.Data data at a scheduled interval from an SNMP MIB table data. |
| System.SnmpQueryProvider | System.Snmp.Library | Outputs System.SnmpData data at a scheduled interval. |
| System.SnmpQuery.FilteredEventProvider | System.Snmp.Library | Outputs System.Event.Data data at a scheduled interval and in accordance with a specified filter expression. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpEventProvider

The System.SnmpEventProvider composite data source module type is used to perform SNMP GET actions at the specified interval, and convert the response data to a System.Event.Data object. The module returns only data that has been successfully retrieved and processed.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Interval | Integer | Contains the interval at which to run the scheduled probe. |
| IsWriteAction | Boolean | Setting this parameter has no effect. The parameter is hard-coded as false. |
| IP | String | Specifies the IP address of the SNMP managed entity. . |
| CommunityString | String | Contains the SNMP Community String.. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| SnmpVarBinds | Complex | Contains a list of SNMP variable bindings as SnmpVarBind elements. |
| EventOriginId | String | Identifies the source of the event by a GUID. The value of this parameter is commonly set to a $MPElement$ variable that contains a GUID that represents the ID of the rule that is running. |
| PublisherId | String | Identifies the event publisher by a GUID. This parameter is commonly set to a $MPElement$ variable that contains a GUID that represents the ID of the rule running. |
| PublisherName | String | Contains the event source. |
| Channel | String | Contains the name of the event log. |
| LoggingComputer | String | Contains the name of the computer the event was logged from. |
| EventNumber | String | Contains the event ID. |
| EventCategory | String | Contains the category of the event that is used to help correlate events to the components of an application. |
| EventLevel | String | Contains the severity level of the event (for example, Information, Warning, or Error). |
| UserName | String | Contains the user name of the user that generated the event. |
| Params | ParamListType | Optional. Specifies one of more event parameters. |

Remarks

The System.SnmpEventProvider data source module comprises three member modules: the [System.Scheduler](#z8d232c0e53514b4c900480d34ddf5a6a) data source module, the [System.SnmpProbe](#za9cf835b51ab4780955b1a384d2ae6f7) probe action module, the [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) module, and [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) condition detection module.

Examples

The following example shows how you can use the System.SnmpEventProvider module in a rule that simply pings all SNMP devices on the network by performing an SNMP GET on the sysName MIB variable. The rule then stores the resulting event data in the Operations Manager database. Note that because the System.SnmpEventProvider module only returns successful pings, unsuccessful pings will not be recorded in the database.

<Rule ID="Microsoft.Samples.Snmp.PingDevice" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Remotable="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="SnmpEventData" TypeID="Snmp!System.SnmpEventProvider">

<Interval>3600</Interval>

<IsWriteAction>false</IsWriteAction>

<IP>$Target/Property[Type="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>public</CommunityString>

<SnmpVarBinds>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.1.5.0</OID>

<Syntax>1</Syntax>

<Value VariantType="8"></Value>

</SnmpVarBind>

</SnmpVarBinds>

<EventOriginId>$Target/Id$</EventOriginId>

<PublisherId>$MPElement$</PublisherId>

<PublisherName>Microsoft Samples</PublisherName>

<Channel>SNMP</Channel>

<LoggingComputer></LoggingComputer>

<EventNumber>0</EventNumber>

<EventCategory>1</EventCategory>

<EventLevel>1</EventLevel>

<UserName/>

<Params/>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="WriteToDB" TypeID="SC!Microsoft.SystemCenter.CollectEvent"/>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpTrap.EventProvider

The System.SnmpTrap.EventProvider composite data source module is used to return SNMP traps as System.Event.Data objects.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| IP | String | Specifies the IP address of the SNMP managed entity from which the traps originate. |
| CommunityString | String | Contains the SNMP Community String. |
| AllTraps | Boolean | Specifies whether all traps from the target device should be sent to Operations Manager. |
| OIDProps | Complex | Contains a list of OIDs to register for traps when AllTraps is set to false. For more information, see the [System.SnmpTrapProvider](#zcbf30137a5e54a8b999cbf97c9685f50) module. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |

Remarks

The System.SnmpTrap.EventProvider data source module comprises two member modules: the [System.Scheduler](#z8d232c0e53514b4c900480d34ddf5a6a) data source module and the [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) condition detection mapping module. The [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) mapping module is configured to return an event that has the following parameters/value pairs:

|  |  |
| --- | --- |
| Parameter | Value |
| EventOriginId | $Target/Id$ |
| PublisherId | $MPElement$ |
| PublisherName | SnmpTrap |
| Channel | SnmpTrap |
| EventNumber | 0 |
| EventCategory | 3 |
| EventLevel | 0 |

To provide filtering for the kind of events you want to generate, see the System.SnmpTrap.FilteredEventProvider, [System.SnmpTrapEventProvider](#z22acfcb9ade14dd58fb27e63faca47db), and [System.SnmpTrapProvider](#zcbf30137a5e54a8b999cbf97c9685f50) module topics.

Examples

The following example shows how you can set up a rule to receive an SNMP cold start trap from SNMP network devices and generate an alert when such a trap is received.

<Rule ID="Microsoft.OM.Samples.Modules.SNMP.ColdStartTrapRule" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" ConfirmDelivery="true" Remotable="true" Priority="Normal" DiscardLevel="100">

<Category>Custom</Category>

<DataSources>

<DataSource ID="EventProvider" TypeID="Snmp!System.SnmpTrap.EventProvider">

<IP>$Target/Property[Type="NetworkDevice!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>$Target/Property[Type="NetworkDevice!Microsoft.SystemCenter.NetworkDevice"]/CommunityString$</CommunityString>

<AllTraps>false</AllTraps>

<OIDProps>

<OIDProp>1.3.6.1.6.3.1.1.5.1</OIDProp>

</OIDProps>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="alert" TypeID="Health!System.Health.GenerateAlert">

<AlertMessageId>$MPElement[Name="Microsoft.ModuleSamples.TrapAlerts.AlertMessage"]$</AlertMessageId>

<Priority>0</Priority>

<Severity>0</Severity>

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpTrapProvider

The System.SnmpTrapProvider data source module type is used to return SNMP traps as System.SnmpData objects.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| IP | String | Specifies the IP address of the SNMP managed entity from which the traps originate. |
| CommunityString | String | Contains the SNMP Community String. |
| AllTraps | Boolean | Specifies whether all traps from the target device should be sent to Operations Manager. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| OIDProps | Complex | Contains a list of OIDs to register for traps when AllTraps is set to false. |

AllTraps

When set to true, the System.SnmpTrap.EventProvider module is configured to send all traps from the targeted device to Operations Manager. When it is set to false, an OIDProps element with at least one OIDProp element must be provided.

<AllTraps>true</AllTraps>

OIDProps

Contains a list of at least one OIDProp element. Each OIDProp element represents an OID that traps from the targeted device will contain. If AllTraps is set to true, this element is not necessary. It should be noted that if this element does contain at least one OIDProp element, AllTraps will be ignored.

<OIDProps>

<OIDProp>1.3.6.1.2.1.4.1.0</OIDProp>

</OIDProps>

Examples

The following example shows how you can set up a rule to receive an SNMP cold start trap from an SNMP network device. When the trap is received, an alert is generated.

<Rule ID="Microsoft.OM.Samples.Modules.SNMP.ColdStartTrapRule2" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" ConfirmDelivery="true" Remotable="true" Priority="Normal" DiscardLevel="100">

<Category>Custom</Category>

<DataSources>

<DataSource ID="trapprovider" TypeID="Snmp!System.SnmpTrapProvider">

<IP>$Target/Property[Type="NetworkDevice!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>$Target/Property[Type="NetworkDevice!Microsoft.SystemCenter.NetworkDevice"]/CommunityString$</CommunityString>

<AllTraps>false</AllTraps>

<OIDProps>

<OIDProp>1.3.6.1.6.3.1.1.5.1</OIDProp>

</OIDProps>

</DataSource>

</DataSources>

<ConditionDetection ID="filter" TypeID="System!System.LogicalSet.ExpressionFilter">

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>ErrorCode</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<XPathQuery>1</XPathQuery>

</ValueExpression>

</SimpleExpression>

</Expression>

<EmptySet>Passthrough</EmptySet>

<SetEvaluation>Any</SetEvaluation>

</ConditionDetection>

<WriteActions>

<WriteAction ID="alert" TypeID="Health!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>0</Severity>

</WriteAction>

</WriteActions>

</Rule>

Related Modules

There are a number of other module types that use this module type as part of a composition that might be more suitable to your scenario. You should use these specialized module types where it is possible rather than declaring your own compositions:

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| System.SnmpTrap.EventProvider | System.Snmp.Library | Used to return SNMP traps as System.Event.Data objects. |
| System.SnmpTrapEventProvider | System.Snmp.Library | Used to return SNMP traps as device-specific System.Event.Data objects. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpTrapEventProvider

The System.SnmpTrapEventProvider composite data source module type is used to return SNMP traps as device specific System.Event.Data objects.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| IP | String | Specifies the IP address of the SNMP managed entity from which the traps originate. |
| CommunityString | String | Contains the SNMP Community String. |
| AllTraps | Boolean | Specifies whether all traps from the target device should be sent to Operations Manager.. For more information, see the [System.SnmpTrapProvider](#zcbf30137a5e54a8b999cbf97c9685f50) module. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| OIDProps | Complex | Contains a list of OIDs to register for traps when AllTraps is set to false. For more information, see the [System.SnmpTrapProvider](#zcbf30137a5e54a8b999cbf97c9685f50) module. |
| EventOriginId | String | Identifies the source of the event by a GUID. The value of this parameter is commonly set to a $MPElement$ variable that contains a GUID that represents the ID of the rule that is running. |
| PublisherId | String | Identifies the event publisher by a GUID. This parameter is commonly set to a $MPElement$ variable that contains a GUID that represents the ID of the rule running. |
| PublisherName | String | Contains the event source. |
| Channel | String | Contains the name of the event log. |
| LoggingComputer | String | Contains the name of the computer the event was logged from. |
| EventNumber | String | Contains the event ID. |
| EventCategory | String | Contains the category of the event that is used to help correlate events to the components of an application. |
| EventLevel | String | Contains the severity level of the event (for example, Information, Warning, or Error). |
| UserName | String | Contains the username of the user that generated the event. |
| Params | ParamListType | Optional. Specifies one of more event parameters. |

Remarks

The System.SnmpTrapEventProvider data source module comprises the following member modules: the [System.Scheduler](#z8d232c0e53514b4c900480d34ddf5a6a) data source module, the [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) module, and the [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) condition detection mapping module.

Examples

The following example shows how you can set up a rule to receive an SNMP cold start trap from SNMP network devices and generate an alert when such a trap is received.

<Rule ID="Microsoft.OM.Samples.Modules.SNMP.ColdStartTrapRule" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" ConfirmDelivery="true" Remotable="true" Priority="Normal" DiscardLevel="100">

<Category>Custom</Category>

<DataSources>

<DataSource ID="EventProvider" TypeID="Snmp!System.SnmpTrap.EventProvider">

<IP>$Target/Property[Type="NetworkDevice!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>$Target/Property[Type="NetworkDevice!Microsoft.SystemCenter.NetworkDevice"]/CommunityString$</CommunityString>

<AllTraps>false</AllTraps>

<OIDProps>

<OIDProp>1.3.6.1.6.3.1.1.5.1</OIDProp>

</OIDProps>

<EventOriginId>$Target/Id$</EventOriginId>

<PublisherId>$MPElement$</PublisherId>

<PublisherName/>

<Channel>System</Channel>

<LoggingComputer/>

<EventNumber>2</EventNumber>

<EventCategory>2</EventCategory>

<EventLevel>1</EventLevel>

<UserName/>

<Params/>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="alert" TypeID="Health!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>0</Severity>

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpTrap.FilteredEventProvider

The System.SnmpTrap.FilteredEventProvider composite data source module type is used to return SNMP traps as a System.Event.Data objects.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| IP | String | Specifies the IP address of the SNMP managed entity from which the traps originate. |
| CommunityString | String | Contains the SNMP Community String. |
| AllTraps | Boolean | Specifies whether all traps from the target device should be sent to Operations Manager. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| OIDProps | Complex | Contains a list of OIDs to register for traps when AllTraps is set to false. |
| Expression | Complex | Contains the expression parameter for its internal [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) condition detection module with which to filter incoming traps. |

AllTraps

When set to true, the System.SnmpTrap.FilteredEventProvider module is configured to send all traps from the targeted device to Operations Manager as event data. When it is set to false, an OIDProps element with at least one OIDProps element must be provided.

<AllTraps>true</AllTraps>

OIDProps

Contains a list of at least one OIDProp element. Each OIDProp element represents an OID that traps from the targeted device will contain. If AllTraps is set to true, this element is not necessary. It should be noted that if this element does contain at least one OIDProp element, AllTraps will be ignored.

<OIDProps>

<OIDProp>1.3.6.1.2.1.4.1.0</OIDProp>

</OIDProps>

Remarks

This composite module contains the following member modules: [System.SnmpTrapProvider](#zcbf30137a5e54a8b999cbf97c9685f50), [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) and [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298). The [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) generates an event data object with the following name/value pairs:

|  |  |
| --- | --- |
| Name | Value |
| PublisherName | SnmpTrap |
| Channel | SnmpTrap |
| LoggingComputer | Empty |
| EventNumber | 0 |
| EventCategory | 3 |
| EventLevel | 0 |
| UserName | Empty |
| Params | Empty |

Examples

The following example shows how you can set up a rule to receive an SNMP cold start trap from a server running a Windows Server operating system and generate an alert.

<Rule ID="Microsoft.Samples.Snmp.DownInterfaceEvent" Accessibility="Internal" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Timeout="300" Remotable="true">

<Category>Custom</Category>

<DataSource ID="SnmpTrapFilteredEventData" TypeID="Snmp!System.SnmpTrap.FilteredEventProvider">

<IP>$Target/Property[Type="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>public</CommunityString>

<AllTraps>false</AllTraps>

<OIDProps>

<OIDProp>1.3.6.1.6.3.1.1.5.1</OIDProp>

</OIDProps>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">ErrorCode</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="UnsignedInteger">1</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</DataSource>

<WriteActions>

<WriteAction ID="Alert" TypeID="SystemHealth!System.Health.GenerateAlert">

<Priority>1</Priority>

<Severity>2</Severity>

<AlertOwner />

<AlertMessageId>$MPElement[Name=Microsoft.Samples.Snmp.ColdBootEvent.TheAlertMessage"]$</AlertMessageId>

<Suppression />

<Custom1 />

<Custom2 />

<Custom3 />

<Custom4 />

<Custom5 />

<Custom6 />

<Custom7 />

<Custom8 />

<Custom9 />

<Custom10 />

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpScanQuery.EventProvider

The System.SnmpScanQuery.EventProvider composite data source module type is used to perform SNMP GET actions in the context of an IP range and at a specified interval. The module returns a System.Event.Data object.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Interval | Integer | Contains the interval at which to get the SNMP data. |
| IPStart | String | Specifies the start address of the IP range. |
| IPEnd | String | Specifies the end address of the IP range. |
| CommunityString | String | Contains the SNMP Community String. |
| Timeout | Unsigned Integer | Specifies the timeout in seconds of the probe action. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| OIDProps | Complex | Contains a list of OIDs upon which to perform the GET action. For more information, see the [System.SnmpScanProbe](#z7f57720b6f0e40839c188942854dbdbe) module. |

Remarks

The System.SnmpScanQuery.EventProvider composite module contains two member modules: the [System.SnmpScanQueryProvider](#zf35161d4a0d94fbb9498016808a28c16) composite data source module and the [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) condition detection module. The [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) module generates a System.Event.Data object with the following parameter values:

|  |  |
| --- | --- |
| Parameter | Value |
| EventOriginId | $Target/Id$ |
| PublisherId | $MPElement$ |
| PublisherName | SnmpScanQuery |
| Channel | SnmpScanQuery |
| LoggingComputer | Empty |
| EventNumber | 0 |
| EventCategory | 3 |
| EventLevel | 0 |
| UserName | Empty |
| Params | Empty |

To implement instance specific parameter values for the returned System.Event.Data object, use the [System.SnmpScanQuery.EventProvider](#z89384c8f1c3f40d3bf58646e1d9d2f5e) module.

To implement a module that returns a System.SnmpData object, use the [System.SnmpScanQueryProvider](#zf35161d4a0d94fbb9498016808a28c16) module.

Examples

The following example illustrates how you can use the System.SnmpScanQuery.EventProvider module in a rule that generates an alert whenever a Microsoft.SystemCenter.NetworkDevice object interface is unavailable over a given IP range.

<Rule ID="Microsoft.Samples.Snmp.DownInterfaceWriteEventProviderScan" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Remotable="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="SnmpEventData" TypeID="Snmp!System.SnmpScanQuery.EventProvider">

<Interval>50000000</Interval>

<IPStart>10.194.229.10</IPStart>

<IPEnd>10.194.229.11</IPEnd>

<CommunityString>public</CommunityString>

<Timeout>7200</Timeout>

<OIDProps>

<OIDProp>1.3.6.1.2.1.2.2.1.8.8</OIDProp>

<OIDProp>.1.3.6.1.2.1.2.2.1.8.9</OIDProp>

</OIDProps>

</DataSource>

</DataSources>

<ConditionDetection ID="Filter" TypeID="System!System.ExpressionFilter">

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[2]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[1]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</ConditionDetection>

<WriteActions>

<WriteAction ID="Alert" TypeID="Health!System.Health.GenerateAlert">

<Priority>1</Priority>

<Severity>2</Severity>

<AlertOwner />

<AlertMessageId>$MPElement[Name="Microsoft.Samples.Snmp.AlertMessage"]$</AlertMessageId>

<Suppression />

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpQueryProvider

The System.SnmpQueryProvider composite data source module type is used to perform SNMP GET actions at the specified interval. The module returns a System.SnmpData object.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Interval | Integer | Contains the interval, in seconds, at which the probe performs the SNMP operation. |
| IsWriteAction | Boolean | Setting this value has no effect. The module only performs GET operations and is therefore hard-coded to false. |
| IP | string | Specifies the IP address of the SNMP managed entity. |
| CommunityString | String | Contains the SNMP Community String. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| SnmpVarBinds | Complex | Contains a list of SNMP variable bindings as SnmpVarBind elements. For more information, see the [System.SnmpProbe](#za9cf835b51ab4780955b1a384d2ae6f7) topic. |

Remarks

The System.SnmpQueryProvider composite module contains two member modules: the [System.SnmpProbe](#za9cf835b51ab4780955b1a384d2ae6f7) probe action module and the [System.Scheduler](#z8d232c0e53514b4c900480d34ddf5a6a) data source module.

If you want to implement an SNMP module that returns a System.Event.Data object, use either the [System.SnmpQueryProvider](#z704961a8e7754ef6b65bc748503b1636) module or the [System.SnmpEventProvider](#z1b2b9e94d10841e7982beee426f526d0) module.

Examples

The following example illustrates how you can use the System.SnmpQueryProvider module in a rule that queries the interface status (the ifOperStatus object) on SNMP devices and creates event data if the interface is down, that is, it returns the value “2”.

<Rule ID="Microsoft.Samples.Snmp.InterfaceData" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Remotable="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="SnmpInterfaceData" TypeID="Snmp!System.SnmpQueryProvider">

<Interval>3600</Interval>

<IsWriteAction>false</IsWriteAction>

<IP>$Target/Property[Type="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>public</CommunityString>

<SnmpVarBinds>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.2.2.1.8.8</OID>

<Syntax>1</Syntax>

<Value VariantType="8"></Value>

</SnmpVarBind>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.2.2.1.8.9</OID>

<Syntax>1</Syntax>

<Value VariantType="8"></Value>

</SnmpVarBind>

</SnmpVarBinds>

</DataSource>

</DataSources>

<ConditionDetection TypeID="System!System.ExpressionFilter" ID="Filter">

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[2]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[1]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</ConditionDetection>

<WriteActions>

<WriteAction ID="GenerateAlert" TypeID="Health!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>0</Severity>

<AlertMessageId>$MPElement[Name="Microsoft.Samples.Snmp.AlertMessage"]$</AlertMessageId>

<Suppression>

<SuppressionValue/>

</Suppression>

</WriteAction>

</WriteActions>

</Rule>

Related Modules

There are a few other module types that use this module type as part of a composition that might be more suitable to your scenario. You should use these specialized module types where it is possible rather than declaring your own compositions:

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| System.SnmpQuery.FilteredEventProvider | System.Snmp.Library | Outputs a templated System.Event.Data object at a scheduled interval from filtered SNMP data. |
| System.SnmpQuery.EventProvider | System.Snmp.Library | Outputs a templated System.Event.Data object at a scheduled interval from SNMP data. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpQuery.FilteredEventProvider

The System.SnmpQuery.FilteredEventProvider composite data source module type is used to perform SNMP GET actions at the specified interval and convert the response data to a System.Event.Data object.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Interval | Integer | Contains the interval, in seconds, at which the probe performs the SNMP operation. |
| IsWriteAction | Boolean | Setting this value has no effect. The module only performs GET operations and is therefore hard-coded to false. |
| IP | String | Specifies the IP address of the SNMP managed.. |
| CommunityString | String | Contains the SNMP Community String. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| SnmpVarBinds | Complex | Contains a list of SNMP variable bindings as SnmpVarBind elements. For more information, see the [System.SnmpProbe](#za9cf835b51ab4780955b1a384d2ae6f7) topic. |
| Expression | Complex | Contains the expression parameter for its internal [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) condition detection module with which to filter the SNMP data. |

Remarks

The System.SnmpQuery.FilteredEventProvider composite module contains the following member modules: the [System.SnmpQueryProvider](#z704961a8e7754ef6b65bc748503b1636) composite data source module, the [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) condition detection module, and [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) condition detection module. The [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) module generates a System.Event.Data object with the following parameter values:

|  |  |
| --- | --- |
| Parameter | Value |
| EventOriginId | $Target/Id$ |
| PublisherId | $MPElement$ |
| PublisherName | SnmpQuery |
| Channel | SnmpQuery |
| LoggingComputer | Empty |
| EventNumber | 0 |
| EventCategory | 3 |
| EventLevel | 0 |
| UserName | Empty |
| Params | Empty |

If you want to implement instance-specific parameter values for the returned System.Event.Data object, instead use the [System.SnmpEventProvider](#z1b2b9e94d10841e7982beee426f526d0) module.

If you want to implement a module that returns a System.SnmpData object, instead use the [System.SnmpQueryProvider](#z704961a8e7754ef6b65bc748503b1636) module.

Examples

The following example illustrates how you can use the System.SnmpQuery.FilteredEventProvider module in a rule that generates an alert whenever a Microsoft.SystemCenter.NetworkDevice object interface is unavailable.

<Rule ID="Microsoft.Samples.Snmp.DownInterfaceWriteEvent" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Remotable="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="SnmpEventData" TypeID="Snmp!System.SnmpQuery.FilteredEventProvider">

<Interval>3600</Interval>

<IsWriteAction>false</IsWriteAction>

<IP>$Target/Property[Type="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice"]/IPAddress$</IP>

<CommunityString>public</CommunityString>

<SnmpVarBinds>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.2.2.1.8.8</OID>

<Syntax>1</Syntax>

<Value VariantType="8"></Value>

</SnmpVarBind>

<SnmpVarBind>

<OID>.1.3.6.1.2.1.2.2.1.8.9</OID>

<Syntax>1</Syntax>

<Value VariantType="8"></Value>

</SnmpVarBind>

</SnmpVarBinds>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[2]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[1]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="Alert" TypeID="Health!System.Health.GenerateAlert">

<Priority>1</Priority>

<Severity>2</Severity>

<AlertOwner />

<AlertMessageId>$MPElement[Name="Microsoft.Samples.Snmp.AlertMessage"]$</AlertMessageId>

<Suppression />

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpScanProbe

The System.SnmpScanProbe probe action module type is used to perform SNMP GET actions in the context of an IP range. The module returns an System.SnmpData data object.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| CommunityString | String | Contains the SNMP Community String . |
| IPStart | String | Specifies the start address of the IP range. |
| IPEnd | String | Specifies the end address of the IP range. |
| OIDProps | Complex | Contains a list of OIDs upon which to perform the GET action. |
| Timeout | Unsigned Integer | Specifies the timeout in seconds of the probe action. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |

OIDProps

Contains a list of at least one OIDProp element. Each OIDProp element represents an OID from which to send traps to Operations Manager.

<OIDProps>

<OIDProp>1.3.6.1.2.1.4.1.0</OIDProp>

</OIDProps>

Remarks

The System.SnmpScanProbe module wraps portions of the WinSNMP API. For more information about how the module works internally, see the [WinSNMP API](http://go.microsoft.com/fwlink/?LinkID=151855&clcid=0x409) topic.

Examples

The following example illustrates how you can use the System.SnmpScanProbe inside a data source module type definition that returns discovery data. The System.SnmpScanProbe module retrieves the SNMP data that is mapped to discovery data by a System.Discovery.FilteredClassSnapshotDataMapper condition detection module that returns discovery data.

<DataSourceModuleType ID="Microsoft.OperationsManager.Samples.SnmpScanProbeDS" Accessibility="Internal" Batching="false">

<Configuration>

<IncludeSchemaTypes>

<SchemaType>System!System.Discovery.MapperSchema</SchemaType>

<SchemaType>System!System.ExpressionEvaluatorSchema</SchemaType>

</IncludeSchemaTypes>

<xsd:element minOccurs="1" name="IPStart" type="xsd:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />

<xsd:element minOccurs="1" name="IPEnd" type="xsd:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />

<xsd:element minOccurs="1" name="CommunityString" type="xsd:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />

<xsd:element minOccurs="1" name="Timeout" type="xsd:unsignedInt" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />

<xsd:element minOccurs="1" name="OIDProps" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType>

<xsd:sequence>

<xsd:element minOccurs="1" maxOccurs="unbounded" name="OIDProp" type="xsd:string" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element minOccurs="1" name="ClassId" type="xsd:string" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />

<xsd:element minOccurs="1" name="InstanceSettings" type="SettingsType" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />

<xsd:element minOccurs="1" name="Expression" type="ExpressionType" xmlns:xsd="http://www.w3.org/2001/XMLSchema" />

</Configuration>

<ModuleImplementation Isolation="Any">

<Composite>

<MemberModules>

<DataSource ID="Scheduler" TypeID="System!System.Scheduler">

<Scheduler>

<SimpleReccuringSchedule>

<Interval Unit="Seconds">41600</Interval>

</SimpleReccuringSchedule>

<ExcludeDates />

</Scheduler>

</DataSource>

<ProbeAction ID="Probe" TypeID="Snmp!System.SnmpScanProbe">

<IPStart>$Config/IPStart$</IPStart>

<IPEnd>$Config/IPEnd$</IPEnd>

<CommunityString>$Config/CommunityString$</CommunityString>

<Timeout>$Config/Timeout$</Timeout>

<OIDProps>$Config/OIDProps$</OIDProps>

</ProbeAction>

<ConditionDetection ID="Mapper" TypeID="System!System.Discovery.FilteredClassSnapshotDataMapper">

<Expression>$Config/Expression$</Expression>

<ClassId>$Config/ClassId$</ClassId>

<InstanceSettings>$Config/InstanceSettings$</InstanceSettings>

</ConditionDetection>

</MemberModules>

<Composition>

<Node ID="Mapper">

<Node ID="Probe">

<Node ID="Scheduler" />

</Node>

</Node>

</Composition>

</Composite>

</ModuleImplementation>

<OutputType>System!System.Discovery.Data</OutputType>

</DataSourceModuleType>

Related Modules

There is one module type that uses this module type as part of a composition that might be more suitable to your scenario. You should use this specialized module type where it is possible rather than declaring your own compositions:

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| System.SnmpScanQueryProvider | System.Snmp.Library | Outputs System.SnmpData data at a scheduled interval across an IP range. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpScanQueryProvider

The System.SnmpScanQueryProvider data source module type is used to perform SNMP GET actions in the context of an IP range and at a specified interval. The module returns a System.SnmpData data object.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Interval | Integer | Contains the interval at which to get the SNMP data. |
| IPStart | String | Specifies the start address of the IP range. |
| IPEnd | String | Specifies the end address of the IP range. |
| CommunityString | String | Contains the SNMP Community String. |
| Timeout | Unsigned Integer | Specifies the timeout in seconds of the probe action. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| OIDProps | Complex | Contains a list of OIDs upon which to perform the GET action. For more information, see the [System.SnmpScanProbe](#z7f57720b6f0e40839c188942854dbdbe) module. |

Remarks

The System.SnmpScanQueryProvider composite module contains two member modules: the [System.SnmpScanProbe](#z7f57720b6f0e40839c188942854dbdbe) probe action module and the [System.Scheduler](#z8d232c0e53514b4c900480d34ddf5a6a) data source module.

To implement an SNMP scan module that returns a System.Event.Data object, use either the [System.SnmpScanQuery.EventProvider](#z89384c8f1c3f40d3bf58646e1d9d2f5e) module or the [System.SnmpScanQuery.FilteredEventProvider](#zca85ace150eb4dc58a18a7426112e9e0) module.

Examples

The following example illustrates how you can use the System.SnmpScanQueryProvider module in a rule that generates an alert whenever a Microsoft.SystemCenter.NetworkDevice object interface is unavailable over a given IP range.

<Rule ID="Microsoft.Samples.Snmp.InterfaceDataScan" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Remotable="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="SnmpInterfaceDataScan" TypeID="Snmp!System.SnmpScanQueryProvider">

<Interval>50000000</Interval>

<IPStart>10.194.229.10</IPStart>

<IPEnd>10.194.229.11</IPEnd>

<CommunityString>public</CommunityString>

<Timeout>7200</Timeout>

<OIDProps>

<OIDProp>1.3.6.1.2.1.2.2.1.8.8</OIDProp>

<OIDProp>.1.3.6.1.2.1.2.2.1.8.9</OIDProp>

</OIDProps>

</DataSource>

</DataSources>

<ConditionDetection TypeID="System!System.ExpressionFilter" ID="Filter">

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[2]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[1]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</ConditionDetection>

<WriteActions>

<WriteAction ID="GenerateAlert" TypeID="Health!System.Health.GenerateAlert">

<Priority>0</Priority>

<Severity>0</Severity>

<AlertMessageId>$MPElement[Name="Microsoft.Samples.Snmp.AlertMessage"]$</AlertMessageId>

<Suppression>

<SuppressionValue/>

</Suppression>

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

System.SnmpScanQuery.FilteredEventProvider

The System.SnmpScanQuery.FilteredEventProvider data source module type is used to perform SNMP GET actions in the context of an IP range and at a specified interval. The module returns a System.Event.Data data object based on the specified expression filter.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| Interval | Integer | Contains the interval at which to get the SNMP data. |
| IPStart | String | Specifies the start address of the IP range. |
| IPEnd | String | Specifies the end address of the IP range. |
| CommunityString | String | Contains the SNMP Community String. |
| Timeout | Unsigned Integer | Specifies the timeout in seconds of the probe action. |
| Version | String | Optional. Specifies the SNMP version. The default value is “2”. To perform an SNMP v1 transaction, the value must be “1”. |
| OIDProps | Complex | Contains a list of OIDs upon which to perform the GET action. For more information, see the [System.SnmpScanProbe](#z7f57720b6f0e40839c188942854dbdbe) module. |
| Expression | Complex | Contains the expression parameter for its internal [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) condition detection module with which to filter the SNMP data. |

Remarks

The System.SnmpScanQuery.FilteredEventProvider composite module contains the following member modules: the [System.SnmpScanQueryProvider](#zf35161d4a0d94fbb9498016808a28c16) composite data source module, the [System.ExpressionFilter](#za762ddc2c8ea4c8e9df078e9074a61ca) condition detection module, and the [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) condition detection module. The [System.Event.GenericDataMapper](#za2276a535a034dd2a205d6316e72e298) module generates a System.Event.Data object with the following parameter values:

|  |  |
| --- | --- |
| Parameter | Value |
| EventOriginId | $Target/Id$ |
| PublisherId | $MPElement$ |
| PublisherName | SnmpScanQuery |
| Channel | SnmpScanQuery |
| LoggingComputer | Empty |
| EventNumber | 0 |
| EventCategory | 3 |
| EventLevel | 0 |
| UserName | Empty |
| Params | Empty |

To implement instance specific parameter values for the returned System.Event.Data object, use the [System.SnmpScanQuery.EventProvider](#z89384c8f1c3f40d3bf58646e1d9d2f5e) module.

To implement a module that returns a System.Snmp.Data object, use the [System.SnmpScanQueryProvider](#zf35161d4a0d94fbb9498016808a28c16) module.

Examples

The following example illustrates how you can use the System.SnmpScanQuery.FilteredEventProvider module in a rule that generates an alert whenever a Microsoft.SystemCenter.NetworkDevice object interface is unavailable over a given IP range.

<Rule ID="Microsoft.Samples.Snmp.DownInterfaceWriteEventScan" Enabled="true" Target="MicrosoftSystemCenterNetworkDeviceLibrary!Microsoft.SystemCenter.NetworkDevice" Remotable="true">

<Category>Custom</Category>

<DataSources>

<DataSource ID="SnmpEventData" TypeID="Snmp!System.SnmpScanQuery.FilteredEventProvider">

<Interval>50000000</Interval>

<IPStart>10.194.229.10</IPStart>

<IPEnd>10.194.229.11</IPEnd>

<CommunityString>public</CommunityString>

<Timeout>7200</Timeout>

<OIDProps>

<OIDProp>1.3.6.1.2.1.2.2.1.8.8</OIDProp>

<OIDProp>.1.3.6.1.2.1.2.2.1.8.9</OIDProp>

</OIDProps>

<Expression>

<And>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[2]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

<Expression>

<SimpleExpression>

<ValueExpression>

<XPathQuery>SnmpVarBinds/SnmpVarBind[1]/Value</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value>2</Value>

</ValueExpression>

</SimpleExpression>

</Expression>

</And>

</Expression>

</DataSource>

</DataSources>

<WriteActions>

<WriteAction ID="Alert" TypeID="Health!System.Health.GenerateAlert">

<Priority>1</Priority>

<Severity>2</Severity>

<AlertOwner />

<AlertMessageId>$MPElement[Name="Microsoft.Samples.Snmp.AlertMessage"]$</AlertMessageId>

<Suppression />

</WriteAction>

</WriteActions>

</Rule>

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Appendix: PowerShell Modules

Microsoft Operations Manager 2007 R2 now includes the ability to run Windows PowerShell scripts from within management packs by using new module types provided in the Windows Core Library management pack. This section covers the capabilities, usage, and troubleshooting of those Windows PowerShell modules.

Important

The Microsoft Operations Manager 2007 agent requires that Windows PowerShell is installed in order to run any Windows PowerShell scripts. Windows PowerShell is not installed by default with the agent and must be installed separately.

What is Windows PowerShell?

Windows PowerShell is a command-line shell and scripting language that can be used to automate many tasks. Windows PowerShell includes small utility programs, called cmdlets, that can either be run directly from the command shell prompt or called from within a batch file or script. Cmdlets can be used by themselves, or they can be combined with other cmdlets to perform complex administrative tasks. Unlike traditional command-line environments that work by returning text results to the end user or routing (“piping”) text to different command-line utilities, Windows PowerShell manipulates .NET Framework objects directly. This provides a more robust and efficient mechanism for interacting with the system.

Note

This topic serves as an overview to the Operations Manager Command Shell. To learn more about Windows PowerShell, see the Windows PowerShell 1.0 Documentation Pack.

Windows PowerShell Hosting in Operations Manager

In previous versions of Operations Manager, it is possible to run Windows PowerShell scripts from management packs by calling the Windows PowerShell executable directly, which often resulted in unacceptable resource utilization. Operations Manager 2007 R2 now provides modules for running Windows PowerShell scripts from within management pack workflows. These scripts are run via hosted Windows PowerShell runspaces. Existing Monitoring Host processes require far less resources than running scripts out of process by using the Windows PowerShell executable.

The Windows PowerShell hosting functionality is available in many workflow types, including discoveries, rules, monitors, agent tasks, diagnostics and recoveries. To enable use this new functionality, a number of new public module types, defined in the Windows Core Library management pack, are provided with Operations Manager 2007 R2 for use by advanced management pack authors. These modules include:

 [Microsoft.Windows.TimedPowerShell.DiscoveryProvider](#z3a8388cf182844d492bd67cf715f40cc)

 [Microsoft.Windows.PowerShellDiscoveryProbe](#z648d3e8e54874d7fa0baab05b11e070a)

 [Microsoft.Windows.PowerShellPropertyBagProbe](#z62e9c3a7f5fa49aaac8dba8da7b3a850)

 [Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe](#zaa5d383a070a479da544600f119e1d12)

 Microsoft.Windows.PowerShellProbe

 Microsoft.Windows.PowerShellWriteAction

 Microsoft.Windows.PowerShellPropertyBagWriteAction

Windows PowerShell Module Inputs

Windows PowerShell modules have a base set of common inputs. This section provides general descriptions of these inputs. For more specific details about syntax and examples, refer to the module documentation.

Windows PowerShell modules recognize the following common parameters:

ScriptName

This parameter is used as an identifier for event logging and tracing purposes. This value is a logic name only and not a file name. Windows PowerShell scripts used by the modules are never stored to the local file system as stand-alone files.

ScriptBody

This parameter contains the actual Windows PowerShell script to be run by the module.

SnapIns

This optional parameter can be used to pass a list of one or more snap-ins (which are used to register sets of cmdlets and providers with Windows PowerShell) that must be loaded within the runspace for the given script. This method of loading snap-ins is preferred over loading the snap-ins from within the Windows PowerShell script because it allows the modules to optimize loading and reuse these snap-ins.

Parameters

Parameters are used to pass information such as static values or variables from runtime into Windows PowerShell scripts, where they can be accessed as variables.

Parameters are defined within the management pack as Name/Value pairs. Each Name/Value pair in the collection is internally added to the $args string array variable in the same order in which they are listed in the collection.

For example, consider the following parameters in this XML code snippet:

<Parameters>

<Parameter>

<Name>stringArg</Name>

<Value>my arg</Value>

</Parameter>

<Parameter>

<Name>intArg</Name>

<Value>5</Value>

</Parameter>

</Parameters>

There are two ways to access these parameter Name/Value pairs in script. The first way is to access the $args string array directly. In the following sample code, the script is accessing the $args elements in order to assign their values to named variables.

$myfirstparam = $args[0]

$mysecondparam = $args[1]

…

The second and easier way to access parameter Name/Value pairs in the script, is to use the Param function. The Param function takes the named variables as arguments. The named variables must be listed in the same order as the list of Name/Value pairs in the collection. The Param function must be the first line in your script. In the following code sample, the script is assigning the $args elements to named variables by simply calling the Param function.

Params($myfirstparam, $mysecondparam)

…

Since Windows PowerShell scripts can access the variables to use the values, it is possible to test the script from a Windows PowerShell command prompt by simply setting the variables and running the script.

The follow table lists examples of possible parameter cases and the behavior for each at script runtime. Note that the variables in the Variable name column are available only if you use one of the above mentioned parameter access methods and explicitly declare their names.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Case | <Name> | <Value> | Variable name | Value | Description |
| Literal string | X | LiteralString | $X | “LiteralString” | The parameter named “X” is assigned a literal string value is accessible in the script as “$X”. |
| <DataItem> XPath query | Y | $Data/EventNumber$ | $Y | XPath result as a string value | At runtime, the XPATH query “$Data/EventNumber$” is replaced with the actual string value and is accessible in the script is “$Y”. |
| Entire <DataItem> | Z | $Data$ | $Z | String equivalent of the entire <DataItem> XML | The entire <DataItem> is passed in the variable “$Z” as an XML string. |

In the following example, two parameters – a string and an integer – are passed to the Windows PowerShell script:

<Task ID="SamplePowerShellTask" Accessibility="Public" Enabled="true" Target="SCLibrary!Microsoft.SystemCenter.HealthService" Timeout="300" Remotable="false">

<Category>Maintenance</Category>

<ProbeAction ID="PA" TypeID="Microsoft.Windows.PowerShellProbe">

<ScriptName>PowerShell Parameter Test Script</ScriptName>

<ScriptBody><![CDATA[

Param($stringArg, $intArg)

]]></ScriptBody>

<Parameters>

<Parameter>

<Name>stringArg</Name>

<Value>

<![CDATA[This is my string arg

with formatting]]></Value>

</Parameter>

<Parameter>

<Name>intArg</Name>

<Value>5</Value>

</Parameter>

</Parameters>

<TimeoutSeconds>60</TimeoutSeconds>

</ProbeAction>

</Task>

You can also pass parameters as a single XML data item, which the script can retrieve by using XPath queries. For example, given the following script definition for the module:

<ScriptBody><![CDATA[

Param($xmlData= [xml]$data)

$xmlData

$xmlData.SelectSingleNode("DataItem/Parameter[1]")

]]></ScriptBody>

<Parameters>

<Parameter>

<Name>data</Name>

<Value>$Data$</Value>

</Parameter>

…the parameter values could extracted using the following command:

PS C:\ > $xmlData.SelectSingleNode("DataItem/Parameter[1]")

…which would produce the following output:

type #text

---- -----

System.String PowerShell is cool

PowerShell Script Output

Windows PowerShell modules can output discovery data, property bags, or serialized .NET objects, depending on the type of module. In both cases, the results will be returned by the script by using the Windows PowerShell Pipeline.Output object.

In the case of a .NET object, the returned object will be interpreted by the Windows PowerShell module into an Operations Manager data type. A single data item which contains a collection of each of the objects returned from the Windows PowerShell script is returned upon completion of the script.

Scalar types are returned as property elements. Complex types are returned as <Object> elements, and any public properties returned as subelements. For example, the following System.String object:

System.String myString = “Sample String”;

…will return the following XML to Operations Manager:

<Property type=’System.String’>Sample String</Property>

Complex types returned by a Windows PowerShell script are presented back to the Operations Manager module as an <Object> element, with any properties on the original complex type being presented as subelements of the <Object> element. For example, this .NET object:

namespace Sample

{

class SampleComplexType

{

public string SampleString { get { return “Sample”; } }

public int SampleInt { get { return 1; } }

public SampleComplexType2 SampleSubType { get { return subtype; } }

public subtype = new SampleComplexType2();

}

class SampleComplexType2

{

public SampleComplexType2() {}

public string SampleString2 { get { return “Sample2”; } }

}

}

…is returned to Operations Manager with the following XML:

<Object type="Sample.SampleComplexType">

<Property name="SampleString" type="System.String">Sample</Property>

<Property name="SampleInt" type="System.Int32">1</Property>

<Object name="SampleSubType" type="System.String">

<Property name="SampleString2" type="System.String">Sample2</Property>

</Object>

</Object>

Scripting Considerations for Operations Manager

This section covers special considerations for Windows PowerShell scripts hosted from within Operations Manager 2007 R2 modules.

Application Isolation and Performance

By default, each script instance will be hosted in a separate runspace in the default application domain. This behavior can be overridden by setting the Isolation registry key to a value of 1.

Application isolation can enhance system stability by preventing simultaneously executed Windows PowerShell scripts and cmdlets – most of which were designed to be called from a single-threaded application – from interfering with each other, but can also cause a severe performance degradation.

To increase performance while running in Isolation mode, AppDomains will be pooled to alleviate the overheard required to instantiate them. Runspaces will not be shared or reused, but they will be cached on each module type.

The Runspace Manager

In order to allow multiple Windows PowerShell scripts to run simultaneously, a mechanism known as a Runspace Manager is used. The Runspace Manager provides a runspace in an isolated application domain in which to run scripts. It implements a First In, First Out (FIFO) queue in which scripts are placed for execution, and handles limiting the number of scripts that are running at the same time.

By default, the maximum number of scripts that can be running in at a time is 20, and scripts in the runspace manager’s queue will expire and be removed after 10 minutes.

Hosting-related Registration Keys

Some default aspects of script execution and behavior in Operations Manager-hosted Windows PowerShell scripts can be customized by setting registry keys. These values can be found in the following node in the registry:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Microsoft Operations Manager\3.0\Modules\Global\PowerShell

The following table shows the registry keys used to control script execution behavior for Windows PowerShell scripts run from an Operations Manager module:

|  |  |  |
| --- | --- | --- |
| Key | Description | Default Value |
| ScriptLimit | Controls how many hosted Windows PowerShell scripts are allowed to run globally. | 0x00000014 |
| QueueMinutes | Defines how many minutes before a script expires from the queue. | 0x0000000a |
| IsolationLevel | Specifies whether a separate AppDomain will be used for each script. A value of 1 indicates that a separate AppDomain is used for each script. | 0x00000000 |

Other Scripting Considerations

Because Windows PowerShell scripts executing from within Operations Manager modules are running in the context of a service, any method or function calls that prompt for user input will throw “not implemented” exceptions. These application programming interfaces (APIs) include:

 PromptForChoice

 PromptForCredential

 ReadLine

 ReadLineAsSecureString

Any method or function calls that are used to write output will be redirected by the Operations Manager host into trace logs that can be used for debugging purposes. For more focused debugging, this output can also be echoed to the ModuleDebug and DbgOut trace logs if the TraceEnabled override is set for the host workflow.

The table below lists the trace level used for each Write API for Windows PowerShell script tracing only (the trace level for Module Debug is always ModuleDebug).

|  |  |
| --- | --- |
| Function | Trace Level |
| WriteDebugLine | Information |
| WriteWarningLine | Warning |
| WriteErrorLine | Error |
| Write | Verbose |
| WriteLine | Verbose |
| WriteVerboseLine | Verbose |

Note

The WriteProgress function is not traced.

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.TimedPowerShell.DiscoveryProvider

The Microsoft.Windows.TimedPowerShell.DiscoveryProvider runs a Windows PowerShell script at a timed interval and returns System.Discovery.Data data.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| IntervalSeconds | string | Specifies, in seconds, how often the Windows PowerShell script runs. Overridable. |
| SyncTime | string | Specifies the time at which the Windows PowerShell script is forced to run. Overridable. |
| ScriptName | String | Specifies the logical name of the Windows PowerShell script to be run. This value will be used in event logs and debug traces to identify the script. |
| ScriptBody | string | Contains the Windows PowerShell script to be run. |
| SnapIns | String | Optional. Contains a list of one or more Windows PowerShell snap-ins that will be pre-loaded for use by the Windows PowerShell script. This method of loading snap-ins is preferred to loading them within the script, as the Operations Manager agent uses this list to optimize loading and unloading of snap-ins. Each snap-in to be loaded must be listed as a distinct child <SnapIn> element within the <SnapIns> element. The SnapIn names used here correspond to the "Name" value returned by the get-pssnapin cmdlet. |
| Parameters | string | Specifies a set of name/value pairs to be used by the Windows PowerShell script as parameters. This value can be a literal string, a $Target reference, a $MPElement reference, or a $Data reference. |
| TimeOutSeconds | integer | Specifies the maximum number of seconds to allow the script to execute before the script is terminated. |
| StrictErrorHandling | Boolean | Optional. Specifies whether to treat script errors as fatal errors instead of warnings. To allow the script error to be raised via the task status, this should always be set to false in the case of discoveries and true-for-task workflows. The default value is false. |

Remarks

For more information and examples on Windows PowerShell script modules, see [Appendix: PowerShell Modules](#zc6b539f2fb5245608af6a09542e6504f).

Examples

The following example demonstrates how to create a discovery that uses the Microsoft.Windows.TimedPowerShell.DiscoveryProvider as a data source. In this example, to create the discovery data item, the script uses the newObject cmdlet with the -ComObject parameter to instantiate the Operations Manager 2007 R2 Scripting Objects.

The script then uses the CreateDiscoveryData, CreateClassInstance, AddProperty, and AddInstance methods from the scripting application programming interfaces (APIs) to construct the discovery data item. At the end of the script, the discovery data is returned simply by calling the variable that contains the discovery data on the last line of the script.

<Discovery ID="Microsoft.Demo.WidgetDiscoveryWithTimedPowerShell" Enabled="true" Target="Windows!Microsoft.Windows.Computer" ConfirmDelivery="false" Remotable="true" Priority="Normal">

<Category>Discovery</Category>

<DiscoveryTypes>

<DiscoveryClass TypeID="Microsoft.Demo.Widget" />

</DiscoveryTypes>

<DataSource ID="TimePowerShell" TypeID="Windows!Microsoft.Windows.TimedPowerShell.DiscoveryProvider">

<IntervalSeconds>43200</IntervalSeconds>

<SyncTime />

<ScriptName>Widget Discovery With TimedPowerShell</ScriptName>

<ScriptBody>

<![CDATA[

# Declare a function that will be used to create a Widget in the specified discovery packet

function CreateWidget

{

Param($Discovery, $ComputerIdentity, $WidgetName)

# Write a statement that could be output by tracing if needed

Write-Verbose "Creating Widget $WidgetName on Computer $ComputerIdentity"

$Instance = $Discovery.CreateClassInstance("$MPElement[Name='Microsoft.Demo.Widget']$")

$Instance.AddProperty("$MPElement[Name='Windows!Microsoft.Windows.Computer']/PrincipalName$", $ComputerIdentity)

$Instance.AddProperty("$MPElement[Name='Microsoft.Demo.Widget']/Key$", $WidgetName)

$Discovery.AddInstance($Instance)

}

# Create a new discovery data packet -- this assumes that the client SDK is available, which it is on all OpsMgr agents

$oAPI = new-object -comObject "MOM.ScriptAPI"

$Discovery = $oAPI.CreateDiscoveryData(0, "$MPElement$", "$Target/Id$")

$ComputerIdentity = "$Target/Property[Type="Windows!Microsoft.Windows.Computer"]/PrincipalName$"

# Create a widget for each subkey under "HKLM\Software\Widgets"

(Get-Item "HKLM:\Software\Widgets").getsubkeynames() | ForEach-Object{ CreateWidget -discovery $Discovery -computerIdentity $ComputerIdentity -widgetName $\_ }

# Return the discovery data by calling the variable

$Discovery

]]>

</ScriptBody>

<TimeoutSeconds>300</TimeoutSeconds>

<StrictErrorHandling />

</DataSource>

</Discovery>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| [Microsoft.Windows.PowerShellDiscoveryProbe](#z648d3e8e54874d7fa0baab05b11e070a) | Microsoft.Windows.Library | Runs a Windows PowerShell script that returns discovery information. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

[Appendix: PowerShell Modules](#zc6b539f2fb5245608af6a09542e6504f)

Microsoft.Windows.PowerShellPropertyBagProbe

Microsoft.Windows.PowerShellPropertyBagProbe probe module runs a Windows PowerShell script to query for information from the system and returns a System.PropertyBag data type. It is defined in the Microsoft.Windows.Library management pack.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ScriptName | string | Specifies the logical name of the Windows PowerShell script to be run. This value will be used in event logs and debug traces to identify the script. |
| ScriptBody | string | Contains the Windows PowerShell script to be run. |
| SnapIns | String | Optional. Contains a list of one or more Windows PowerShell snap-ins that will be pre-loaded for use by the Windows PowerShell script. This method of loading snap-ins is preferred to loading them within the script, as the Operations Manager agent uses this list to optimize loading and unloading of snap-ins. Each snap-in to be loaded must be listed as a distinct child <SnapIn> element within the <SnapIns> element. The SnapIn names used here correspond to the "Name" value returned by the get-pssnapin cmdlet. |
| Parameters | string | Specifies a set of Name/Value pairs to be used by the Windows PowerShell script as parameters. This value can be a literal string, a $Target reference, a $MPElement reference, or a $Data reference. |
| TimeOutSeconds | integer | Specifies the maximum number of seconds to allow the script to execute before the script is terminated. |
| StrictErrorHandling | Boolean | Optional. Specifies whether to treat script errors as fatal errors instead of warnings. This should always be set to false in the case of discoveries, monitors, and diagnostics, and true for task workflows. The default value is false. |

Remarks

This module can be used in a monitor to run a Windows PowerShell script to determine state information about a monitored computer and return a name-value collection as a property bag.

Examples

In the following section of example code, a Windows PowerShell script is used as part of a probe to retrieve a value from the registry, add the value as a property named “State” to a property bag object, and return the property bag object to the module.

From there, the value of the returned property is checked as part of the monitor’s ErrorExpression and SuccessExpression configuration to determine the monitor’s state.

The following code shows the definition of the monitor:

<UnitMonitorType ID="Microsoft.Demo.PowerShellModule.Widget.PowerShellModule.TwoStateMonitorType.PropertyBag" Accessibility="Public">

<MonitorTypeStates>

<MonitorTypeState ID="Error" NoDetection="false"/>

<MonitorTypeState ID="Success" NoDetection="false"/>

</MonitorTypeStates>

<Configuration>

<IncludeSchemaTypes>

<SchemaType>System!System.ExpressionEvaluatorSchema</SchemaType>

<SchemaType>Windows!Microsoft.Windows.PowerShellSchema</SchemaType>

</IncludeSchemaTypes>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="IntervalSeconds" type="xsd:integer"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SyncTime" type="xsd:string"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="ScriptName" type="xsd:string"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="ScriptBody" type="xsd:string"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SnapIns" type="SnapInsType" minOccurs="0" maxOccurs="1"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Parameters" type="NamedParametersType" minOccurs="0" maxOccurs="1"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="TimeoutSeconds" type="xsd:integer"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="StrictErrorHandling" type="xsd:boolean" minOccurs="0" maxOccurs="1"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SerializationDepth" type="xsd:integer" minOccurs="0" maxOccurs="1"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="ErrorExpression" type="ExpressionType"/>

<xsd:element xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SuccessExpression" type="ExpressionType"/>

</Configuration>

<OverrideableParameters>

<OverrideableParameter ID="IntervalSeconds" Selector="$Config/IntervalSeconds$" ParameterType="int"/>

<OverrideableParameter ID="SyncTime" Selector="$Config/SyncTime$" ParameterType="string"/>

<OverrideableParameter ID="TimeoutSeconds" Selector="$Config/TimeoutSeconds$" ParameterType="int"/>

</OverrideableParameters>

<MonitorImplementation>

<MemberModules>

<DataSource ID="DataSource" TypeID="System!System.SimpleScheduler">

<IntervalSeconds>$Config/IntervalSeconds$</IntervalSeconds>

<SyncTime>$Config/SyncTime$</SyncTime>

</DataSource>

<ProbeAction ID="Script" TypeID="Windows!Microsoft.Windows.PowerShellPropertyBagProbe">

<ScriptName>$Config/ScriptName$</ScriptName>

<ScriptBody>$Config/ScriptBody$</ScriptBody>

<SnapIns>$Config/SnapIns$</SnapIns>

<Parameters>$Config/Parameters$</Parameters>

<TimeoutSeconds>$Config/TimeoutSeconds$</TimeoutSeconds>

<StrictErrorHandling>$Config/StrictErrorHandling$</StrictErrorHandling>

<SerializationDepth>$Config/SerializationDepth$</SerializationDepth>

</ProbeAction>

<ConditionDetection ID="ErrorFilter" TypeID="System!System.ExpressionFilter">

<Expression>$Config/ErrorExpression$</Expression>

</ConditionDetection>

<ConditionDetection ID="SuccessFilter" TypeID="System!System.ExpressionFilter">

<Expression>$Config/SuccessExpression$</Expression>

</ConditionDetection>

</MemberModules>

<RegularDetections>

<RegularDetection MonitorTypeStateID="Error">

<Node ID="ErrorFilter">

<Node ID="Script">

<Node ID="DataSource"/>

</Node>

</Node>

</RegularDetection>

<RegularDetection MonitorTypeStateID="Success">

<Node ID="SuccessFilter">

<Node ID="Script">

<Node ID="DataSource"/>

</Node>

</Node>

</RegularDetection>

</RegularDetections>

</MonitorImplementation>

</UnitMonitorType>

The next example shows the monitor being used with a Windows PowerShell script:

<UnitMonitor ID="Microsoft.Demo.PowerShellModule.Widget.PowerShellModule.CheckWidgetStatusMonitor.PropertyBag"

Enabled="true"

Accessibility="Internal"

ConfirmDelivery="false" Target="Microsoft.Demo.PowerShellModule.Widget.PowerShellModule" TypeID="Microsoft.Demo.PowerShellModule.Widget.PowerShellModule.TwoStateMonitorType.PropertyBag"

ParentMonitorID="Health!System.Health.EntityState">

<Category>StateCollection</Category>

<AlertSettings AlertMessage="Microsoft.Demo.PowerShellModule.Widget.PowerShellModule.CheckWidgetStatusMonitor.PropertyBag\_AlertMessageResourceID">

<AlertOnState>Error</AlertOnState>

<AutoResolve>false</AutoResolve>

<AlertPriority>Normal</AlertPriority>

<AlertSeverity>Error</AlertSeverity>

</AlertSettings>

<OperationalStates>

<OperationalState HealthState="Success" MonitorTypeStateID="Success" ID="GreenState" />

<OperationalState HealthState="Error" MonitorTypeStateID="Error" ID="RedState" />

</OperationalStates>

<Configuration>

<IntervalSeconds>15</IntervalSeconds>

<SyncTime/>

<ScriptName>Monitor Widgets</ScriptName>

<ScriptBody>

<![CDATA[

# Always use the param statement to easily

# access parameters

param($SleepyTime)

# Get access to the scripting API

$oAPI = new-object -comObject "MOM.ScriptAPI"

# Create the property bag

$pb = $oAPI.CreatePropertyBag()

# Populate the property bag with data from the registy

$pb.AddValue("State",

(get-itemproperty hklm:/software/widgets/$Target/Property[Type='Microsoft.Demo.PowerShellModule.Widget.PowerShellModule']/Key$).State)

$pb

Write-Host "Sleeping $SleepyTime seconds starting"

Start-Sleep $SleepyTime

]]>

</ScriptBody>

<Parameters>

<Parameter>

<Name>SleepyTime</Name>

<Value>5</Value>

</Parameter>

</Parameters>

<TimeoutSeconds>120</TimeoutSeconds>

<ErrorExpression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">Property[@Name='State']</XPathQuery>

</ValueExpression>

<Operator>NotEqual</Operator>

<ValueExpression>

<Value Type="String">good</Value>

</ValueExpression>

</SimpleExpression>

</ErrorExpression>

<SuccessExpression>

<SimpleExpression>

<ValueExpression>

<XPathQuery Type="String">Property[@Name='State']</XPathQuery>

</ValueExpression>

<Operator>Equal</Operator>

<ValueExpression>

<Value Type="String">good</Value>

</ValueExpression>

</SimpleExpression>

</SuccessExpression>

</Configuration>

</UnitMonitor>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe | Microsoft.Windows.Library | A trigger-only module that runs a Windows PowerShell script to query for information from the system and returns a System.PropertyBag data type. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe

Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe trigger-only probe module runs a Windows PowerShell script to query for information from the system and returns a System.PropertyBag data type. It is defined in the Microsoft.Windows.Library management pack.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ScriptName | string | Specifies the logical name of the Windows PowerShell script to be run. This value will be used in event logs and debug traces to identify the script. |
| ScriptBody | string | Contains the Windows PowerShell script to be run. |
| SnapIns | String | Optional. Contains a list of one or more Windows PowerShell snap-ins that will be pre-loaded for use by the Windows PowerShell script. This method of loading snap-ins is preferred to loading them within the script, as the Operations Manager agent uses this list to optimize loading and unloading of snap-ins. Each snap-in to be loaded must be listed as a distinct child <SnapIn> element within the <SnapIns> element. The SnapIn names used here correspond to the "Name" value returned by the get-pssnapin cmdlet. |
| Parameters | string | Specifies a set of name/value pairs to be used by the Windows PowerShell script as parameters. This value can be a literal string, a $Target reference, a $MPElement reference, or a $Data reference. |
| TimeOutSeconds | integer | Specifies the maximum number of seconds to allow the script to execute before the script is terminated. |
| StrictErrorHandling | Boolean | Optional. Specifies whether to treat script errors as fatal errors instead of warnings. This should always be set to false in the case of discoveries, monitors, and diagnostics, and true for task workflows. The default value is false. |

Remarks

This module can be used in on-demand detection scenarios; for example, when a user starts a task. This module has its TriggerOnly property set true. If you want a probe module that uses input data, use the [Microsoft.Windows.PowerShellPropertyBagProbe](#z62e9c3a7f5fa49aaac8dba8da7b3a850) module instead.

Examples

In the following section of example code, a Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe module is used to run a Windows PowerShell script that displays a list of PIDs and Handle counts for all svcHost.exe running processes.

<Task ID="Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbeTask" Accessibility="Public" Enabled="true" Target="Windows!Microsoft.Windows.Computer" Timeout="300">

<Category>Custom</Category>

<ProbeAction ID="Probe" TypeID="Windows!Microsoft.Windows.PowerShellPropertyBagTriggerOnlyProbe">

<ScriptName>Shows a list of PIDS and Handle Counts for all "svcHost" processes</ScriptName>

<ScriptBody>

param($ProcessName)

# Create a new property bag

$oAPI = new-object -comObject "MOM.ScriptAPI"

$pb = $oAPI.CreatePropertyBag()

get-process $ProcessName | %{$pb.AddValue($\_.Id,$\_.Handles)}

$pb

</ScriptBody>

<Parameters>

<Parameter>

<Name>ProcessName</Name>

<Value>svchost</Value>

</Parameter>

</Parameters>

<TimeoutSeconds>180</TimeoutSeconds>

<StrictErrorHandling>true</StrictErrorHandling>

</ProbeAction>

</Task>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| Microsoft.Windows.PowerShellPropertyBagProbe | Microsoft.Windows.Library | Runs a Windows PowerShell script to query for information from the system and returns a System.PropertyBag data type. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

Microsoft.Windows.PowerShellDiscoveryProbe

Microsoft.Windows.PowerShell.DiscoveryProbe probe action module allows Windows PowerShell script to gather and return discovery data. This module is available in the Microsoft.Windows.Library management pack.

Parameters

This module takes the following configuration parameters:

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| ScriptName | string | Specifies the logical name of the Windows PowerShell script to be run. This value will be used in event logs and debug traces to identify the script. |
| ScriptBody | string | Contains the Windows PowerShell script to be run. |
| SnapIns | Complex | Optional. Contains a list of one or more Windows PowerShell snap-ins that will be pre-loaded for use by the Windows PowerShell script. This method of loading snap-ins is preferred to loading them within the script, as the Operations Manager agent uses this list to optimize loading and unloading of snap-ins. Each snap-in to be loaded must be listed as a distinct child <SnapIn> element within the <SnapIns> element. The SnapIn names used here correspond to the "Name" value returned by the get-pssnapin cmdlet. |
| Parameters | string | Specifies a set of Name/Value pairs to be used by the Windows PowerShell script as parameters. This value can be a literal string, a $Target reference, a $MPElement reference, or a $Data reference. |
| TimeOutSeconds | integer | Specifies the maximum number of seconds to allow the script to execute before the script is terminated. |
| StrictErrorHandling | Boolean | Optional. Specifies whether to treat script errors as fatal errors instead of warnings. This should always be set to false in the case of discoveries and should only be true for task workflows, to allow the script error to be raised via the task status. The default value is false. |

Remarks

The Microsoft [Microsoft.Windows.TimedPowerShell.DiscoveryProvider](#z3a8388cf182844d492bd67cf715f40cc) module provides the necessary functionality of running a Windows PowerShell script on a timed interval.

Because the Microsoft.Windows.PowerShellDiscoveryProbe module is only intended to be used as a part of a custom composite module type definition, in most cases, the parameters of this module will actually be references to details passed in via the module’s configuration, rather than literal values.

For more information and examples on Windows PowerShell script modules, see [Appendix: PowerShell Modules](#zc6b539f2fb5245608af6a09542e6504f).

Examples

The following example shows how this module is used in the definition of the Microsoft.Windows.TimedPowerShell.DiscoveryProvider data source to act a reusable provider of discovery data from Windows PowerShell scripts that are run on a regular schedule:

<

<DataSourceModuleType ID="Microsoft.Windows.TimedPowerShell.DiscoveryProvider" Accessibility="Public">

<Configuration>

<IncludeSchemaTypes>

<SchemaType>Microsoft.Windows.PowerShellSchema</SchemaType>

</IncludeSchemaTypes>

<xsd:element name="IntervalSeconds" type="xsd:int" />

<xsd:element name="SyncTime" type="xsd:string" />

<xsd:element name="ScriptName" type="NonNullString" />

<xsd:element name="ScriptBody" type="NonNullString" />

<xsd:element name="SnapIns" type ="SnapInsType" minOccurs="0" maxOccurs="1"/>

<xsd:element name="Parameters" type="NamedParametersType" minOccurs="0" maxOccurs="1"/>

<xsd:element name="TimeoutSeconds" type="xsd:integer" />

<xsd:element name="StrictErrorHandling" type="xsd:boolean" minOccurs="0" maxOccurs="1"/>

</Configuration>

<OverrideableParameters>

<OverrideableParameter ID="IntervalSeconds" Selector="$Config/IntervalSeconds$" ParameterType="int" />

<OverrideableParameter ID="SyncTime" Selector="$Config/SyncTime$" ParameterType="string" />

<OverrideableParameter ID="TimeoutSeconds" Selector="$Config/TimeoutSeconds$" ParameterType="int" />

</OverrideableParameters>

<ModuleImplementation>

<Composite>

<MemberModules>

<DataSource TypeID="System!System.Discovery.Scheduler" ID="DS1">

<Scheduler>

<SimpleReccuringSchedule>

<Interval>$Config/IntervalSeconds$</Interval>

<SyncTime>$Config/SyncTime$</SyncTime>

</SimpleReccuringSchedule>

<ExcludeDates />

</Scheduler>

</DataSource>

<ProbeAction TypeID="Microsoft.Windows.PowerShellDiscoveryProbe" ID="PSScript">

<ScriptName>$Config/ScriptName$</ScriptName>

<ScriptBody>$Config/ScriptBody$</ScriptBody>

<SnapIns>$Config/SnapIns$</SnapIns>

<Parameters>$Config/Parameters$</Parameters>

<TimeoutSeconds>$Config/TimeoutSeconds$</TimeoutSeconds>

<StrictErrorHandling>$Config/StrictErrorHandling$</StrictErrorHandling>

</ProbeAction>

</MemberModules>

<Composition>

<Node ID="PSScript">

<Node ID="DS1" />

</Node>

</Composition>

</Composite>

</ModuleImplementation>

<OutputType>System!System.Discovery.Data</OutputType>

</DataSourceModuleType>

Related Modules

|  |  |  |
| --- | --- | --- |
| Module Type | Defined In | Usage |
| [Microsoft.Windows.TimedPowerShell.DiscoveryProvider](#z3a8388cf182844d492bd67cf715f40cc) | Microsoft.Windows.Library | Runs a Windows PowerShell script at a timed interval that returns discovery information. |

See Also

[Operations Manager 2007 R2 Management Pack Module Reference](#z95f75fb1626a4474b1bca93aa3f385be)

[Appendix: PowerShell Modules](#zc6b539f2fb5245608af6a09542e6504f)