

# Sample MP

---

## Event Based Collection Rules

**Steve Wilson**

**3/26/2007**

This guide details the contents of the sample event collection rule management pack (AuthorMPs.Demo.EventCollectionRules).

## Contents

Introduction .....	3
Discovery .....	3
Views .....	3
Rules.....	4
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectEvent101.....	4
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectEvent102.....	4
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectEvent103.....	4
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectMultipleEventsUsingRegex.....	5
AuthorMPs.Demo.EventCollectionRules.ApplicationXComponent.CollectEvent301 .....	5
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectServiceChangeEvents .....	6
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectErrorEventsFromTextLog.....	6
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectErrorEventsFromMultipleTextLogs.....	7
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectWarningFromParam2InCSVLog .....	7
AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectAllEventsInCSVLog .....	8
AuthorMPs.Demo.EventCollectionRules.ApplicationXComponent.CollectSpecificComponentEvents....	8

## Introduction

This management pack shows you how to define collection rules from the following data sources

- Windows Event
- WMI Event
- Generic Text Log
- Generic Delimited Text Log

In addition it shows the following concepts:

- Simple Expressions
- Regular Expressions
- Writing to the Operations Database and / or the Data Warehouse
- Using Target properties as criteria

## Discovery

The MP defines two classes:

- AuthorMPs Demo - Application X
- AuthorMPs Demo - Application X Component

To discover instances of these classes, create a directory called C:\AuthorMPs (this will create an instance of Application X). Now create some files in that directory (this will create one instance of Application X Component for each file).

## Views

There are four views in this management pack:

- Application X State - a state view showing all instances of Application X
- Application X Events - an event view showing all events stored for all instances Application X
- Application X Component State - a state view showing all instances of Application X Component
- Application X Component Events - an event view showing all events stored for all instances of Application X Component

You should note that the event view for Application X will also show events stored against the Applicaiton X components.

As well as using the views I have provided, you can pivot from a state view to an event view. For example, to see events for a specific Application X, find the instance you want in the state view then right click and select Open, Event View. This will show you events just for this specific instance.

If you want to check which rule was responsible for a collected event you can pivot to the rule from the event details view.

## Rules

Here is a summary of the rules in the management pack and how you can see them working:

### [AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectEvent101](#)

**Target: Application X**

This is a simple Windows event collection rule. It shows the use of a simple expression that looks at the event publisher (EventCreate) and the Event ID (101). To store this event use the eventcreate utility that is part of Windows Server 2003 or later. At a command prompt on a server that you have a discovered instance of Application X type:

```
EventCreate /ID 101 /T information /D Test
```

This rule will store the event in both the Operations Database and the Data Warehouse - this is done by having two write actions.

### [AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectEvent102](#)

**Target: Application X**

This is another simple Windows event collection rule that looks for event 102 from the EventCreate source. The difference to the 101 collection rule is that this rule only writes data to the Data Warehouse. If you fire this event you will not see the event in the Operations Console, only with an event report against the warehouse.. To generate the event type:

```
EventCreate /ID 102 /T information /D Test
```

### [AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectEvent103](#)

**Target: Application X**

This is another simple Windows Event collection rule that looks for event 103 from the EventCreate source. The rule adds one additional criterion to look for the text **Test** in parameter 1 of the event. EventCreate stores the whole description as parameter 1. To see an event that matches the expression type:

```
EventCreate /ID 103 /T information /D Test
```

To see an event that does not match type:

```
EventCreate /ID 103 /T information /D Foo
```

## AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectMultipleEventsUsingRegex

### Target: Application X

This rule again uses the EventCreate source but this time I use a regular expression to look for multiple events. Any event from the EventCreate source with an event ID of 200, 201, 202 or 203 will be stored. Any of the following will create an event that is stored:

```
EventCreate /ID 200 /T information /D Test
```

```
EventCreate /ID 201 /T information /D Test
```

```
EventCreate /ID 202 /T information /D Test
```

```
EventCreate /ID 203 /T information /D Test
```

## AuthorMPs.Demo.EventCollectionRules.ApplicationXComponent.CollectEvent301

### Target: Application X Component

This rule introduces a new concept. This rule uses a \$Target notation in part of the expression:

```
<Expression>
  <SimpleExpression>
    <ValueExpression>
      <XPathQuery Type="String">Params/Param[1]</XPathQuery>
    </ValueExpression>
    <Operator>Equal</Operator>
    <ValueExpression>
      <Value
Type="String">$Target/Property[Type="AuthorMPs.Demo.EventCollectionRules.Appl
icationXComponent"]/ID$</Value>
    </ValueExpression>
  </SimpleExpression>
</Expression>
```

This is an important concept. There are multiple instances of the Application X Component discovered (if you created multiple files in the C:\AuthorMPs directory). A workflow defined for this rule will run against each instance of the Application X Component. If I did not use some property of the Application X Component as the criteria I would get multiple copies of the event stored. When I specify the above I am saying “only store the event for a specific Application X Component if the event has parameter 1 equal to the ID property of the instance”.

To show this working, suppose I have 2 instances of Application X Component with ID of “Comp1” and “Comp2”. If you type the following at a command prompt:

EventCreate /ID 301 /T information /D Comp1

This will store an event against the Comp1 instance – you can see this by pivoting to an event view for the Comp1 instance. If you pivot from any other instance of Application X Component you will not see an event.

EventCreate /ID 301 /T information /D Comp2

This will store an event against the Comp1 instance – you can see this by pivoting to an event view for the Comp2 instance.

Now try to log an event for an ID that does not exist e.g. Assume I do not have an instance with an ID of Comp3. Type:

EventCreate /ID 301 /T information /D Comp3

You should not see this event written to the database – you can check by looking at the Application X Component event view or the Application X event view.

## **AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectServiceChangeEvents**

### **Target: Application X**

This rule shows the collection of WMI events. This particular rule looks for Windows service modification events e.g. start or stop of a service.

Notice I use the following syntax for the namespace:

```
$Target/Host/Property[Type="Windows!Microsoft.Windows.Computer"]/NetworkName$\root\cimv2
```

This ensures that the rule would work agentlessly by substituting in the correct computer name.

To see an event stored for this rule, stop any service on a monitored server.

## **AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectErrorEventsFromTextLog**

### **Target: Application X**

This rule shows collection from a single text log. Each line of the log is checked for a match against the expression. Since this is a non delimited log file the entire line is passed as parameter 1 in the event (Params/Param[1] in XPath terms). See how I use this to look for the Error: expression in the text somewhere.

This rule uses the following log file:

```
C:\AuthorMPsLogs\ AppXGeneric.log
```

To collect an event from this file type the following at a command prompt:

```
Echo Error: This is a test >> C:\AuthorMPsLogs\ AppXGeneric.log
```

You should see an event under Application X Events. Note that if the log does not exist when you first imported the management pack, the first time you log to the file it may take a few minutes for Operations Manager to pick up the event you logged.

To see that a line without the text Error: in does not get stored type:

```
Echo Info: This is a test >> C:\AuthorMPsLogs\ AppXGeneric.log
```

## AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectErrorEventsFromMultipleTextLogs

### Target: Application X

This is a similar rule to the previous rule. The difference is that it uses a wild card to specify the log file so events will be stored from multiple files. This rule looks for files AppX\*.txt in the directory C:\AuthorMPsLogs. As with the previous rule it will store events where there is the text Error: in the line.

Try creating a couple of log files that match the wildcard and log some events. Note there may be a short delay the first time the log file is created.

## AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectWarningFromParam2InCSVLog

### Target: Application X

This rule shows you how to collect events from a generic text log file that is delimited by some character (in this case I use a comma). When you collect from a delimited file each field will be collected as a parameter. Therefore if you have three fields and a delimiter comma consider the following log file entry:

```
Some field 1, some field 2, some field3
```

This would be stored in event parameters as:

```
<Params>  
  <Param>Some field 1</Param>  
  <Param>Some field 2</Param>  
  <Param>Some field 3</Param>  
</Params>
```

So you would use the XPath Params/Param[1], Params/Param[2] or Params/Param[3] to access the fields in your expression module.

In my example rule I look for the text Error: in the second field. I look in the log file C:\AuthorMPsLogs\ AppXGenericCSV.log. To generate a log entry that is collected type:

```
Echo Some Field 1, Error:Some Error, Some Field 3 >> C:\AuthorMPsLogs\ AppXGenericCSV.log
```

If you generated the following log entry it would not be stored

```
Echo Error:Some Error, Some Field 2, Some Field 3 >> C:\AuthorMPsLogs\ AppXGenericCSV.log
```

## **AuthorMPs.Demo.EventCollectionRules.ApplicationX.CollectAllEventsInCSVLog**

### **Target: Application X**

This rule is similar to the previous rule and collects events from a comma separated log file. However, this time there is an empty expression which means all events are stored. You should be careful with empty criteria as this could potentially cause a large number of events to be stored in Operations Manager.

This rule uses the following log: C:\AuthorMPsLogs\ AppXGenericCSV2.log.

## **AuthorMPs.Demo.EventCollectionRules.ApplicationXComponent.CollectSpecificComponentEvents**

### **Target: Application X Component**

This rule is another delimited log file collection rule. This time it uses a \$Target property for the expression in parameter 1:

```
$Target/Property[Type="AuthorMPs.Demo.EventCollectionRules.ApplicationXComponent"]/ID$
```

This means only events specific to the instance will be collected.

To show this working, suppose I have 2 instances of Application X Component with ID of "Comp1" and "Comp2". If you type the following at a command prompt:

```
Echo Comp1,Some Field 2,some Field 3 >> C:\AuthorMPsLogs\ AppXComponentGenericCSV.log
```

This will store an event against the Comp1 instance – you can see this by pivoting to an event view for the Comp1 instance. If you pivot from any other instance of Application X Component you will not see an event. If you do not have an instance called Comp3, the following log file entry will not get stored:

```
Echo Comp3,Some Field 2,some Field 3 >> C:\AuthorMPsLogs\ AppXComponentGenericCSV.log
```