Non-SAP Monitoring using OS scripts for multiple metrics

With SAP Solution Manager 7.1 SP12

Introduction: This guide describes how you can set up and us OS scripts to monitor SAP and non-SAP applications with SAP Solution Manager 7.1 Monitoring and Alerting Infrastructure. With this new version of the OS script monitoring you are now able to run scripts that return more than one metric in one script run. This way you can save resources (in form of script executions) on the managed system.

Contents

WHAT'S NEW

With the first version of the OS script monitoring the script could only return exactly one metric with each script run. This led to a high amount of script executions and added load to the managed system.

With this new version of the script data provide it is possible to return more than one metric with one script run. The data provider can return a formatted set of metrics which then can be interpreted by the data collector in Solution Manager and be consumed by monitoring.

Restrictions

There are some restrictions when using this new data collector. The restrictions are due to the monitoring infrastructure and how data collector templates are defined. If you want to use this data provider you have to create one dedicated data collector template for each script with each parameter you want to run. Furthermore it is recommended to group metrics in a script that are collected with the same collection frequency.

PREREQUISITES

SAP Solution Manager Support Package Level

Make sure your SAP Solution Manager runs at least on SAP Solution Manager 7.1 SP12.
LM-Service needs to be deployed with at least the following patch levels:
Solution Manager 7.1 SP12 → LM-Service SP12 patch 04
Solution Manager 7.1 SP13 → LM-Service SP13 patch 02
Solution Manager 7.1 SP14 and higher → already part of the initial SP delivery (patch 00)

Install Diagnostics Agent and SAP Hostagent

The OS script is triggered and the result are consumed locally by the Diagnostics agent that is installed on the host for the 3rd party application. Therefore it is necessary to install a Diagnostics agent on the 3rd party host.

Please refer to SAP note 1448655 for the installation guide for the Diagnostics agent. The SAP Hostagent is usually installed together with the Diagnostics agent. If you want to install the SAP Hostagent separately please refer to SAP note 1031096.

Create a technical system in Solution Manager

If your managed system is a non-SAP system and doesn't have SLD data suppliers that make sure it is created in LMDB automatically, you will have to create a technical system in LMDB manually. How to create a non-SAP system in LMDB is described in the guide "Creating Unspecific Cluster Systems for TechMon and BPMon in LMDB" under http://wiki.scn.sap.com/wiki/display/TechOps/System+Monitoring+-+How-to+Guides

CREATING THE OS SCRIPT

For this guide we created a Microsoft Windows PowerShell script which returns several performance metrics for a process. For the detailed script please refer to the appendix of this document.

The PowerShell script is started by a batch script which can be run by the diagnostics agent. The batch script hands over the process name, which will be the input parameter for the metric as well and some other PowerShell related parameters to start the script correctly.

The return values have to follow a certain syntax to be interpretable by the data provider. The return set contains of several lines with the syntax described below. For each metric you need one line.

```
<metric type="<metric type>" name="<metric name>"
[path=""<param1>=<value1>[|<param2>=<value2>]""] value="""$objItem.Name"""
[rating=""0|1|2|3""]/>
```

With the following fields:

- type (mandatory) = type of the metric, can be string, integer or float
- name (mandatory) = name of the metric, will be used to pick the correct metric later on during the custom metric creation in MAI

- path (optional) = for metric groups to build a metric path if more than one value is returned for one metric (e.g. if more than one OS process matches the search pattern)
- value (mandatory) = the measured value, must match the return type
- rating (optional) = for already rated metrics, can be 0=grey, 1=green, 2=yellow or 3=red

So you can have basically four combinations for the return values:

1. **Name and Value**
   This is for simple metrics that do not use the threshold type “Already rated”
   Example: `<metric type=""string"" name=""ProcessName"" value="""$objItem.Name"""/>`

2. **Name, Value and Rating**
   This is for simple metrics that use the threshold type “Already rated”
   Example: `<metric type=""string"" name=""Availability"" value=""Process is available"" rating=""1""/>`

3. **Name, Path and Value**
   This in for metric groups that do not use the threshold type “Already rated”
   Example: `<metric type=""integer"" name=""ThreadCount"" path=""PROCNAME="$objItem.Name""" value="""$objItem.ThreadCount"""/>`

4. **Name, Path, Value and Rating**
   This in for metric groups that use the threshold type “Already rated”
   Example: `<metric type=""integer"" name=""ThreadCount"" path=""PROCNAME="$objItem.Name""" value="""$objItem.ThreadCount""" rating=""1""/>`

Please note that option 1 and 2 only work for simple metrics but not for metric groups.

If you test your script in a shell, the output would have to look like this:

```
C:\usr\sap\custom_scripts>PowershellProcessMonMulti.bat
<metric type="string" name="ProcessName" value="saphostexec"/>
<metric type="integer" name="CreatingProcessID" path="PROCNAME=saphostexec" value="504"/>
<metric type="integer" name="ElapsedTime" path="PROCNAME=saphostexec" value="4603125"/>
<metric type="integer" name="HandleCount" path="PROCNAME=saphostexec" value="151"/>
<metric type="integer" name="IDProcess" path="PROCNAME=saphostexec" value="1880"/>
<metric type="integer" name="IODataBytesPersec" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="IODataOperationsPersec" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="IOOtherBytesPersec" path="PROCNAME=saphostexec" value="1256"/>
<metric type="integer" name="IOOtherOperationsPersec" path="PROCNAME=saphostexec" value="78"/>
<metric type="integer" name="IOReadBytesPersec" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="IOReadOperationsPersec" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="IOWriteBytesPersec" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="IOWriteOperationsPersec" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="PageFaultsPersec" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="PageFileBytes" path="PROCNAME=saphostexec" value="15527936"/>
<metric type="integer" name="PageFileBytesPeak" path="PROCNAME=saphostexec" value="15691776"/>
<metric type="integer" name="PercentPrivilegedTime" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="PercentProcessorTime" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="PercentUserTime" path="PROCNAME=saphostexec" value="0"/>
<metric type="integer" name="PoolNonpagedBytes" path="PROCNAME=saphostexec" value="10432"/>
<metric type="integer" name="PoolPagedBytes" path="PROCNAME=saphostexec" value="111432"/>
<metric type="integer" name="PriorityBase" path="PROCNAME=saphostexec" value="8"/>
<metric type="integer" name="PrivateBytes" path="PROCNAME=saphostexec" value="15527936"/>
<metric type="integer" name="ThreadCount" path="PROCNAME=saphostexec" value="6"/>
<metric type="integer" name="VirtualBytes" path="PROCNAME=saphostexec" value="201064448"/>
<metric type="integer" name="VirtualBytesPeak" path="PROCNAME=saphostexec" value="234758144"/>
<metric type="integer" name="WorkingSet" path="PROCNAME=saphostexec" value="6332416"/>
<metric type="integer" name="WorkingSetPeak" path="PROCNAME=saphostexec" value="13594624"/>
<metric type="string" name="Availability" value="Process is available." rating="1"/>
<metric type="integer" name="ProcessCount" value="1"/>
```

Please note that it doesn’t matter if you use a bash, PowerShell, Perl or shell script. Everything works as long as the output is formatted correctly.

SETTING UP THE OS SCRIPT AND THE DIAGNOSTICS AGENT

Placing the OS Script on the host

To be able to use the OS script you have to place it in the folder `/usr/sap/custom_scripts` on Unix or in `<drive>:\usr\sap\custom_scripts` on Windows.

The file type has to be either a windows batch file .bat or on Unix an executable shell script .sh (don't forget to set it to executable).

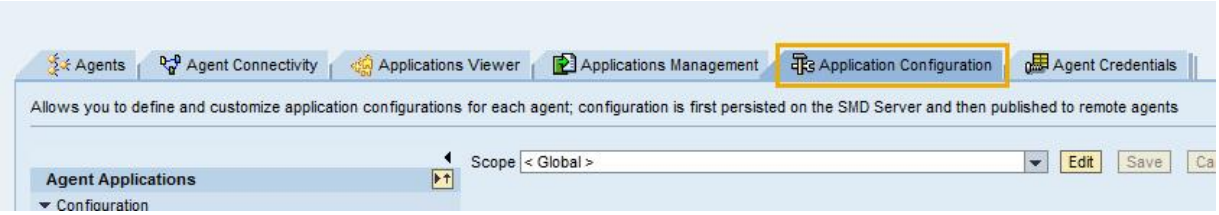
Adjustments to the Diagnostics Agent Configuration

To enable the diagnostics agent to find the file, you have to add the custom_scripts folder location to the commands.xml in the diagnostics agent configuration.

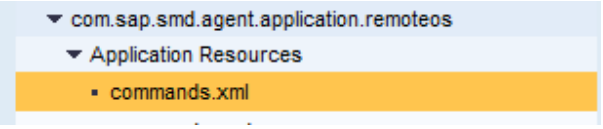
Go to the Diagnostics Agent Administration, e.g. using the URL:

http(s)://<solman-host>:<solman-port>/smd/AgentAdmin

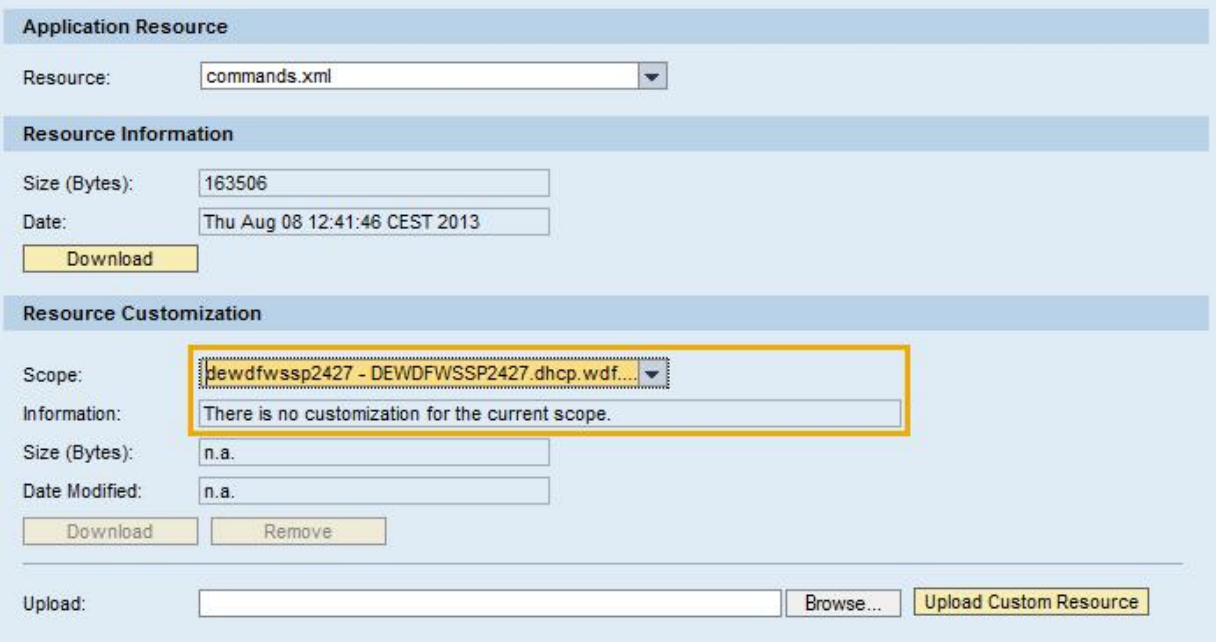
Switch to the tab "Application Configuration"



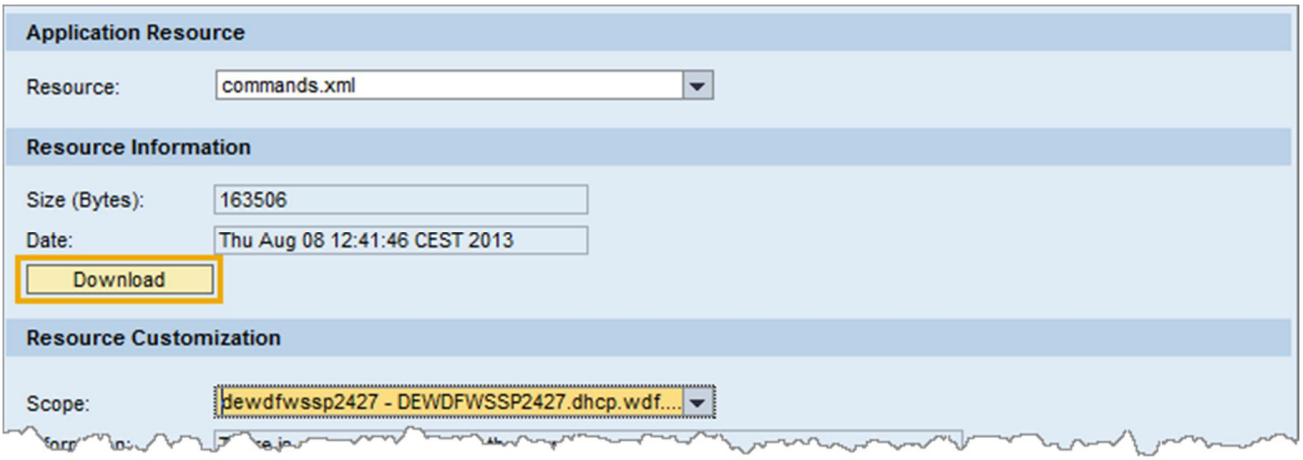
In the left hand navigation tree navigate to com.sap.smd.agent.application.remoteos → Application Resources → commands.xml



Check if there was already customizing done for this resource for the agent you plan to use for the non-SAP application. Select the agent in the scope selection.



If there is any customizing done, download the customized file from the agent. This way you make sure you don't overwrite existing customizing. If no customizing exists for this agent download the default resource using the Download button above.



Open the xml with an XML editor and adjust the following lines. The content already exists in the default XML, but you have to adjust the paths.

```
<CmdGroup name="Custom scripts" cv_ppms_id="*">
  <Cmd key="os.run" name="Script" desc="Execute scripts from a standard directory.">
    <OsCmd ostype="WINDOWS" exec="" path="" param="false" runtime="600"/>
    <OsCmd ostype="UNIX" exec="" path="" param="false" runtime="600"/>
  </Cmd>
</CmdGroup>
```

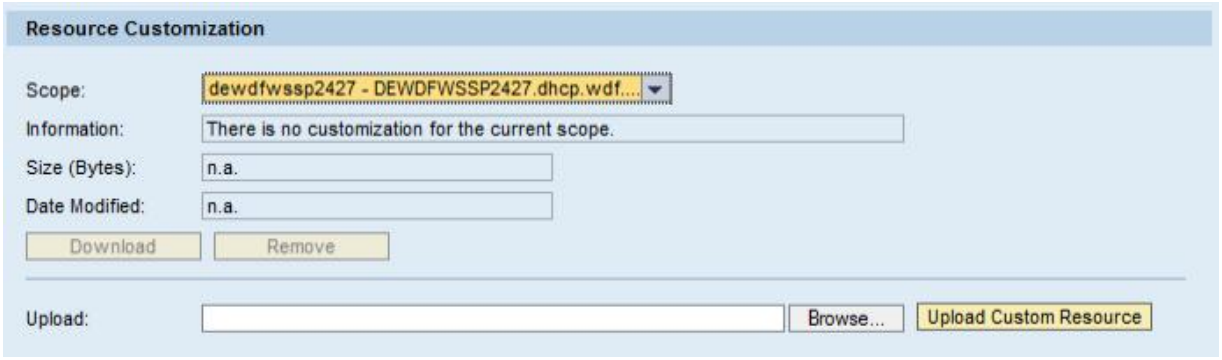
Change it to:

```
<CmdGroup name="Custom scripts" cv_ppms_id="*">
  <Cmd key="os.run" name="Script" desc="Execute scripts from a standard directory.">
    <OsCmd ostype="WINDOWS" exec="" path="C:/usr/sap" param="true" runtime="600"/>
    <OsCmd ostype="UNIX" exec="" path="" param="false" runtime="600"/>
  </Cmd>
</CmdGroup>
```

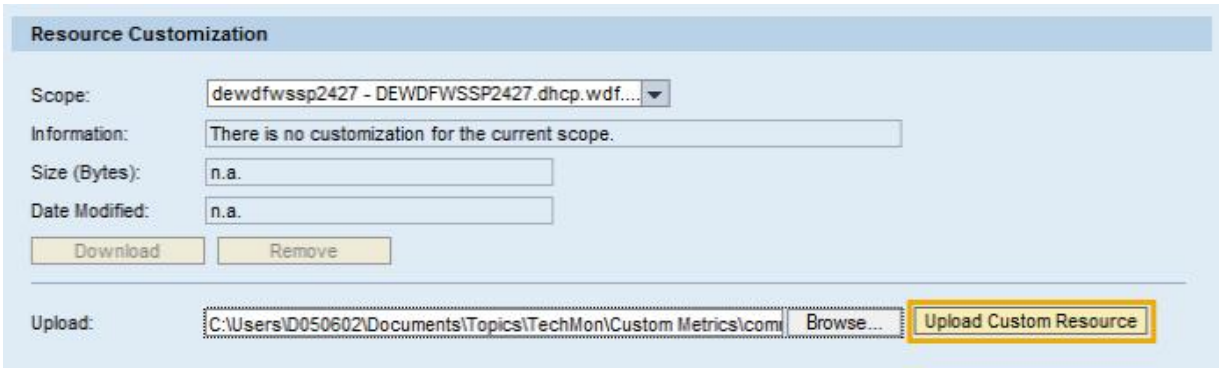
Make sure you use forward slashes also for Windows paths.

If you want to use OS scripts on Windows and Unix and are sure to always use the same path and windows drive, you can maintain the entry for Unix as well and upload the command.xml as new global resource. In our example we will only upload it for our one agent. In our example we will only upload it for our one agent.

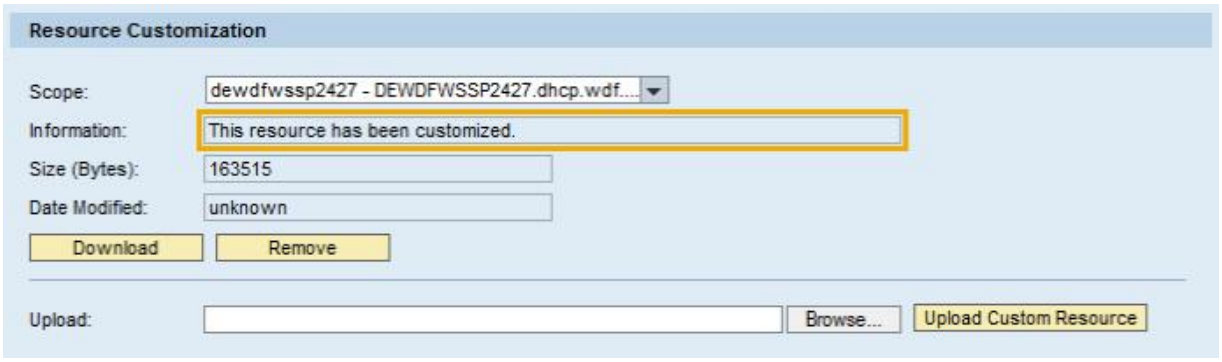
After adjusting and saving the changed file go back to the agent administration. Select the agent for which you want to upload the file as scope.



Browse for your commands.xml file and upload it to the agent.



After the successful upload you can see that the resource has been customized.



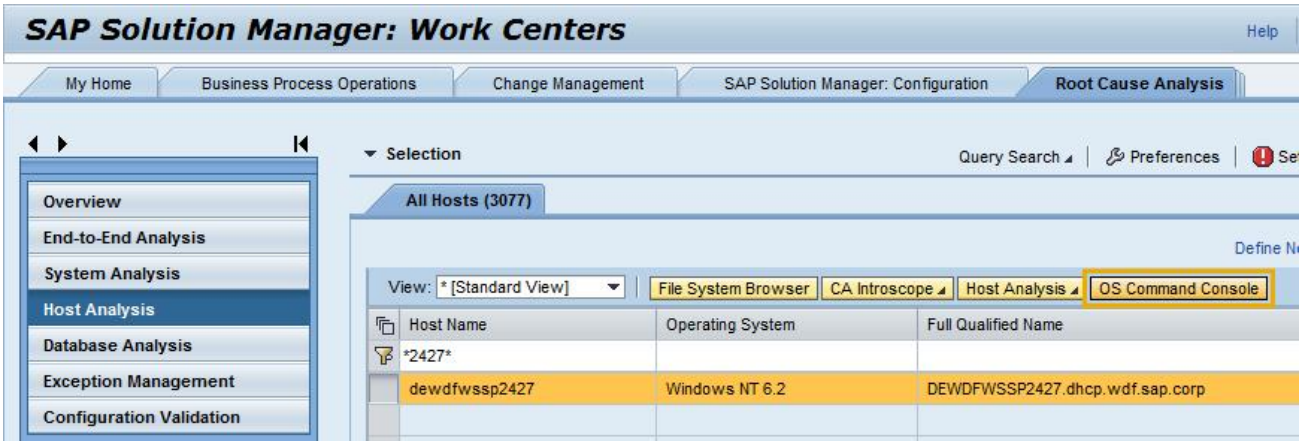
You have to restart the diagnostics agent for the changes to take effect. Also if you place a new script in the custom_scripts folder you have to restart the diagnostics agent.

Testing the Script from Solution Manager

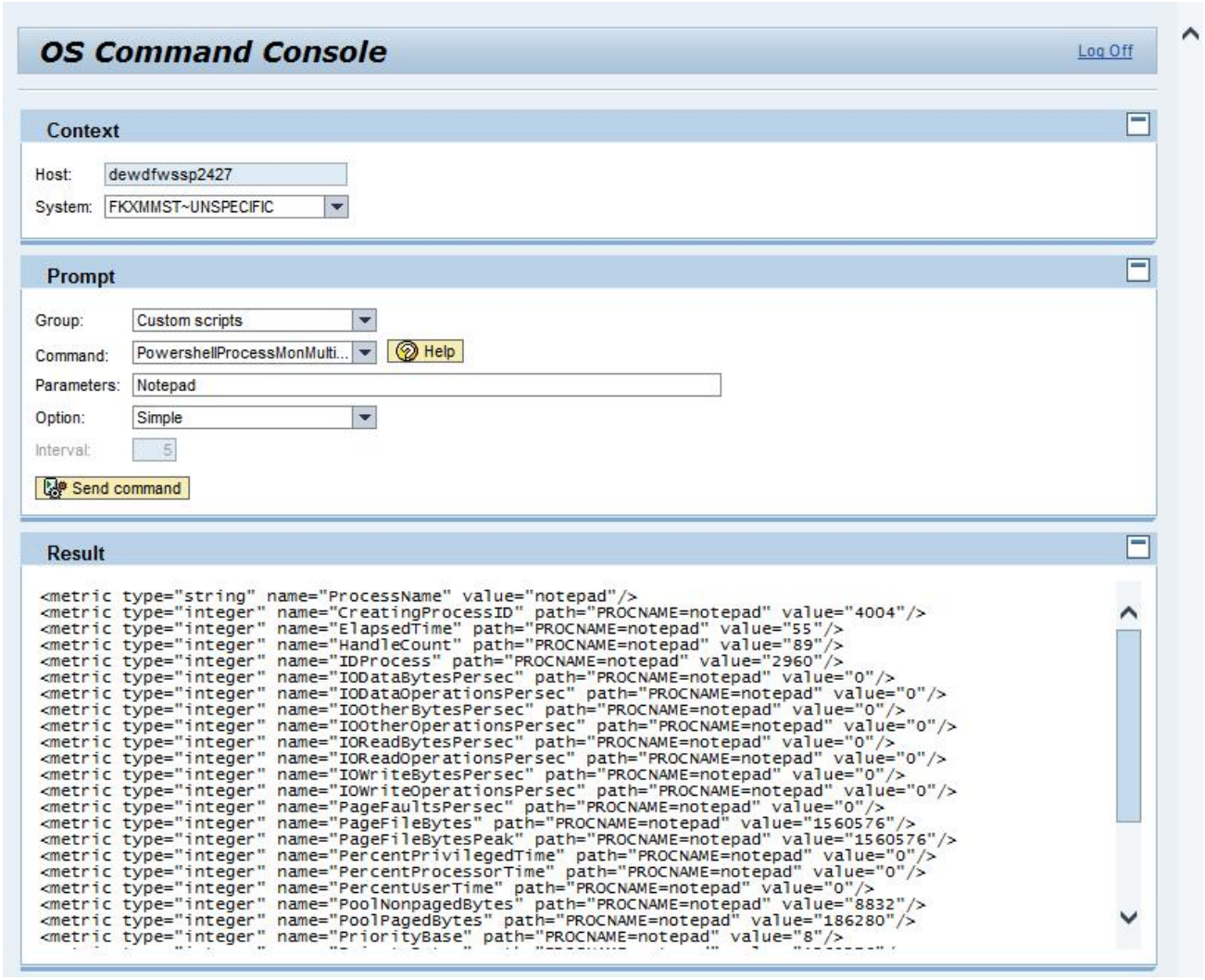
You can test if the script can be executed by the Diagnostics agent by running it from the OS Command Console in Solution Manager.

Call transaction SM_WORKCENTER and switch to the Root Cause Analysis workcenter.

Go to “Host Analysis”, select your host and open the OS Command Console:



In the OS Command Console you now select the group “Custom scripts” and as command the batch script that starts the PowerShell script. In the field parameters you can send the process name.



CREATING A DATA COLLECTOR TEMPLATE

The next step is now to create a custom data collector for the script.

Right now you have to create one data collector for each script with each parameter. The reason is that the script name and the script parameter has to be a collector context parameter and these can only be handed over directly to the data collector. We know this is unfortunate but right now the infrastructure leaves no other way.

The simplest data collector will only contain the following parameters:

- SCRIPT (Mandatory, hidden, type CIP): The name of the script you want to run (in our case the batch script that starts the PowerShell)
- OPTION (Mandatory, hidden, type CIP): The process name you want to monitor
- NAME (Config, type MP, no ranges, no wildcards): The name of the metric you want from the return set

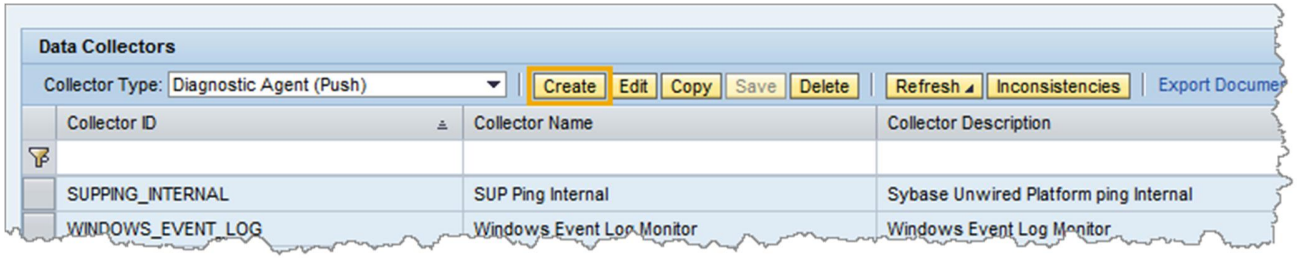
The parameter NAME is set when you create the custom metric.

Note that this kind of data collector will only support single metrics without the *path* tag in the return value. This data collector cannot be used to build metric groups.

To create the data collector go to the Data Collector maintenance UI:

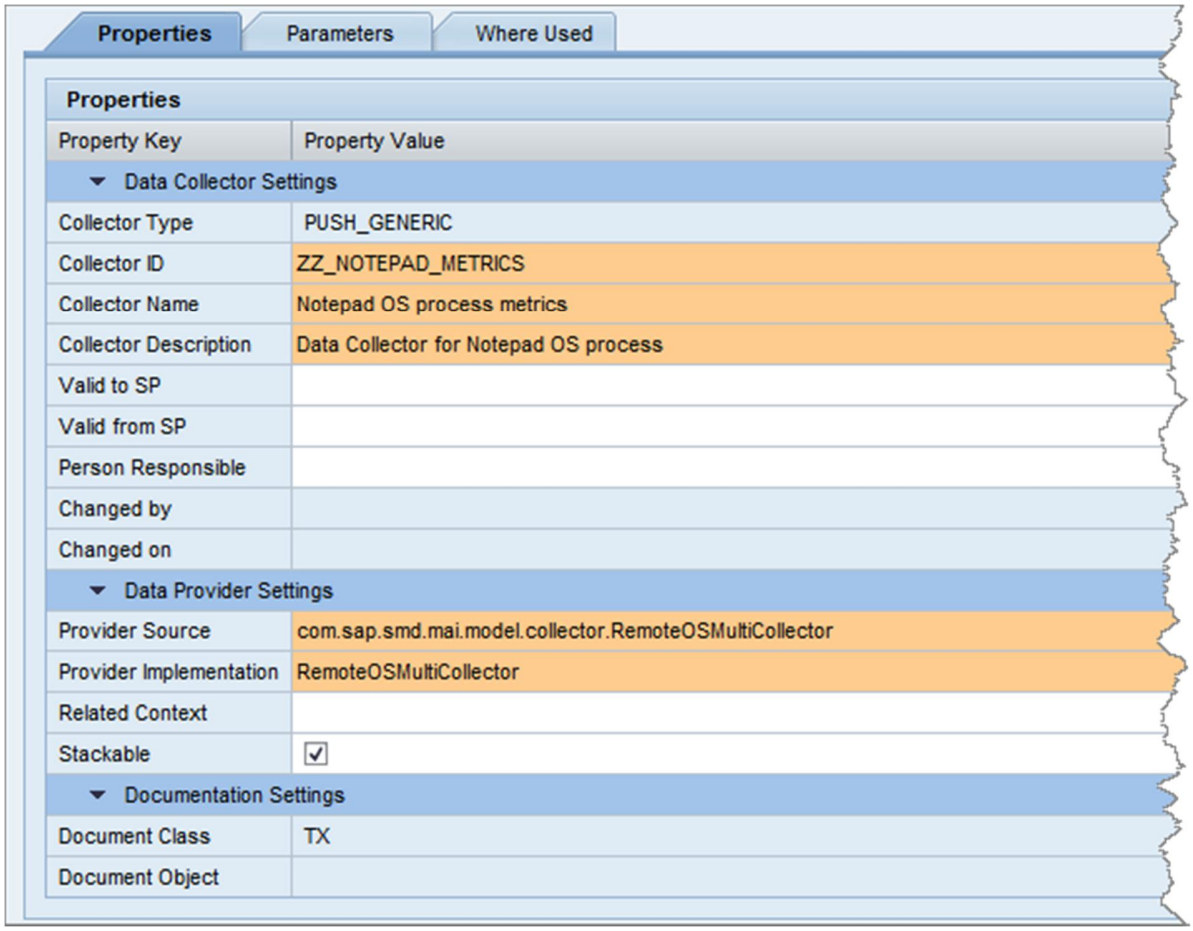
http://<solman host>:<solman port>/sap/bc/webdynpro/sap/wd_mai_dpc_main

Here switch to the Collector Type “Diagnostic Agent (Push)” and click the “Create” Button.



Enter the Data Collector properties.

- Collector ID: The technical name for the collector: Make sure the collector ID starts with ZZ, otherwise the collector will be overwritten with the next content update.
- Collector Name: The name of the data collector, you will see this in the technical monitoring setup
- Collector Description: A description
- Provider Source: com.sap.smd.mai.model.collector.RemoteOSMultiCollector
- Provider Implementation: RemoteOSMultiCollector

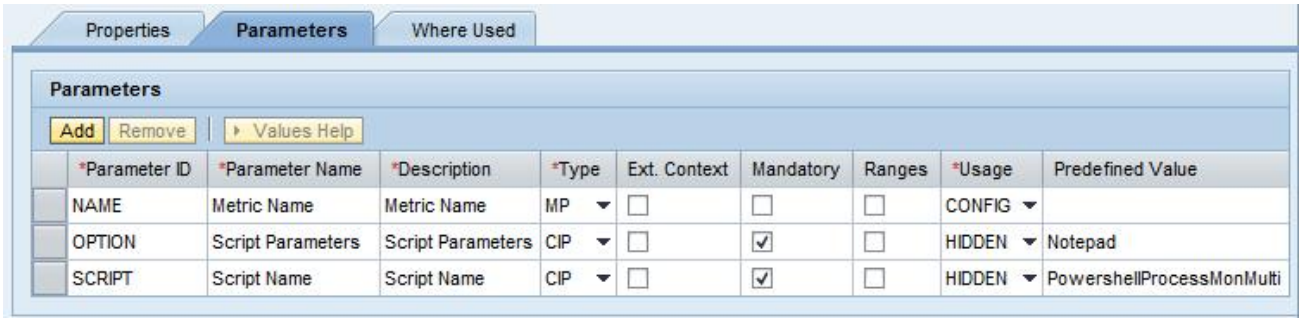


Properties | Parameters | Where Used

Properties

Property Key	Property Value
▼ Data Collector Settings	
Collector Type	PUSH_GENERIC
Collector ID	ZZ_NOTEPAD_METRICS
Collector Name	Notepad OS process metrics
Collector Description	Data Collector for Notepad OS process
Valid to SP	
Valid from SP	
Person Responsible	
Changed by	
Changed on	
▼ Data Provider Settings	
Provider Source	com.sap.smd.mai.model.collector.RemoteOSMultiCollector
Provider Implementation	RemoteOSMultiCollector
Related Context	
Stackable	☑
▼ Documentation Settings	
Document Class	TX
Document Object	

On the next tab create the necessary parameters. Make sure you select the correct type and usage.



Properties | Parameters | Where Used

Parameters

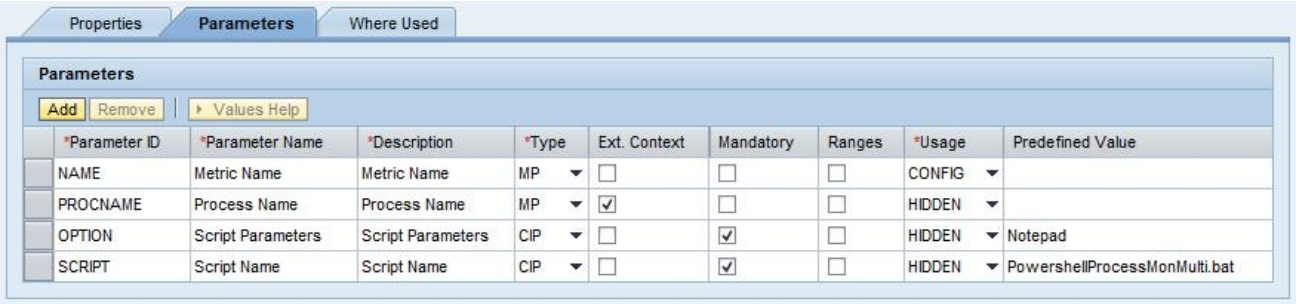
Add | Remove | Values Help

*Parameter ID	*Parameter Name	*Description	*Type	Ext. Context	Mandatory	Ranges	*Usage	Predefined Value
NAME	Metric Name	Metric Name	MP	☐	☐	☐	CONFIG	
OPTION	Script Parameters	Script Parameters	CIP	☐	☑	☐	HIDDEN	Notepad
SCRIPT	Script Name	Script Name	CIP	☐	☑	☐	HIDDEN	PowershellProcessMonMulti

The parameters mist be named exactly as shown above. Make sure you use the full name (incl. extension) for the script name. Otherwise it won't work and you will spend some time with troubleshooting …

If you have more sophisticated return values that contain a path to build metric groups you have to add each variable in the metric path as metric parameter to the data collector. Please note that these parameters cannot be used to filter on the result set. The only filter parameter is NAME, for the metric name. The other parameters are necessary to be able to build a metric group. Hence it makes sense to maintain them as HIDDEN, so they are not visible during the metric setup.

In our case only PROCNAME is a path parameter. Hence we add this to the data collector. Make sure you select "Ext. Context" for every path parameter.



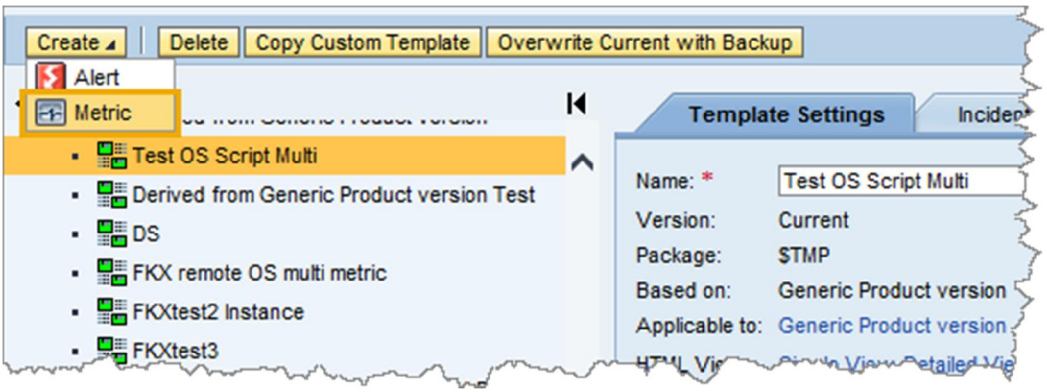
*Parameter ID	*Parameter Name	*Description	*Type	Ext. Context	Mandatory	Ranges	*Usage	Predefined Value
NAME	Metric Name	Metric Name	MP	☐	☐	☐	CONFIG	
PROCNAME	Process Name	Process Name	MP	☑	☐	☐	HIDDEN	
OPTION	Script Parameters	Script Parameters	CIP	☐	☑	☐	HIDDEN	Notepad
SCRIPT	Script Name	Script Name	CIP	☐	☑	☐	HIDDEN	PowershellProcessMonMulti.bat

Now our data collector template is complete and we can start building the custom metric.

CREATE CUSTOM METRIC AND ALERT

For our example we create a new template on instance level. We use the template type "Generic Product Version". At first we create a custom template.

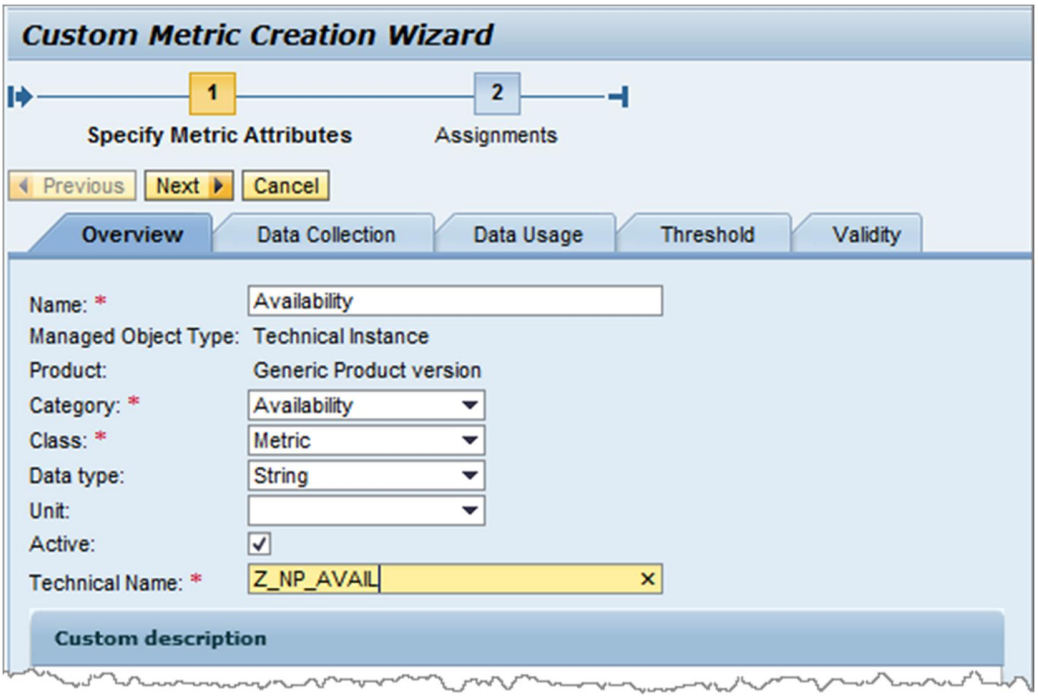
Switch to "Expert Mode" by clicking the respective button in the upper right corner, to be able to create a custom metric. Click the "Create" button to create the metric.



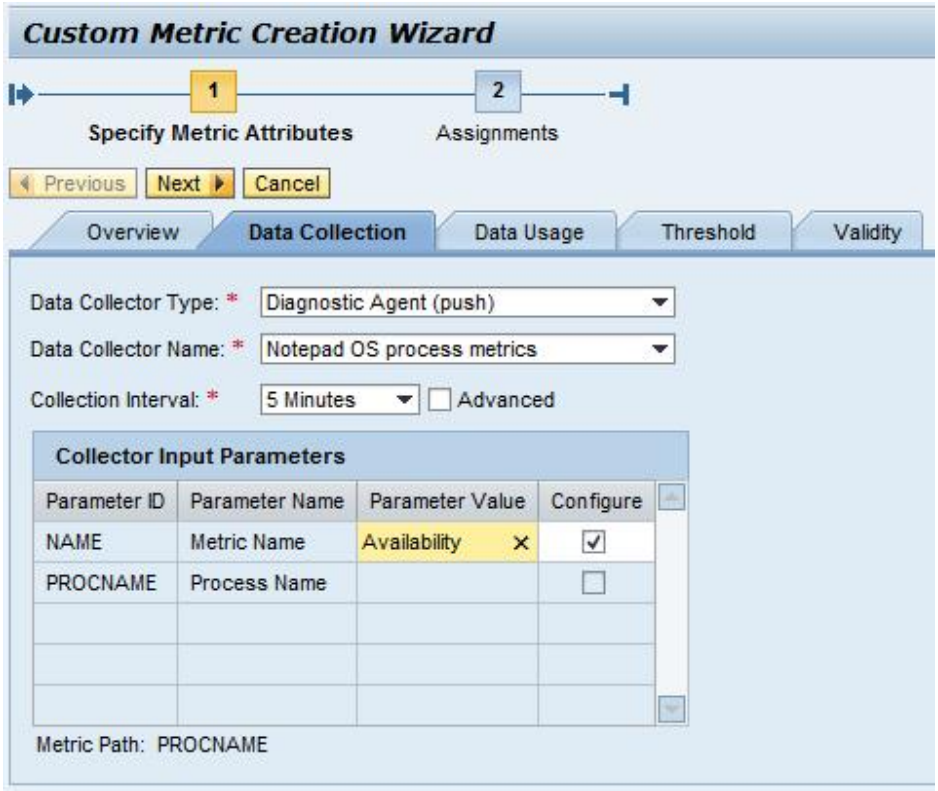
The first metric is a metric without path in the return value, it is also already rated.

```
<metric type=""string"" name=""Availability"" value=""Process is available matching the
regular expression "$ProcessName""" rating=""1""/>
```

On the first tab you have to maintain some metric properties. Make sure you select "Metric" for the class and "String" for the data type. Otherwise it won't work.

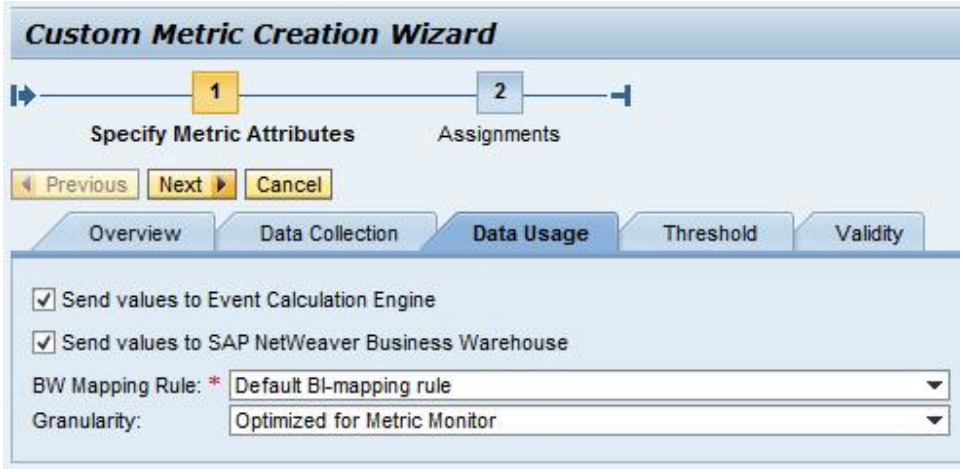


On the next tab you select the new data collector and enter the metric name.

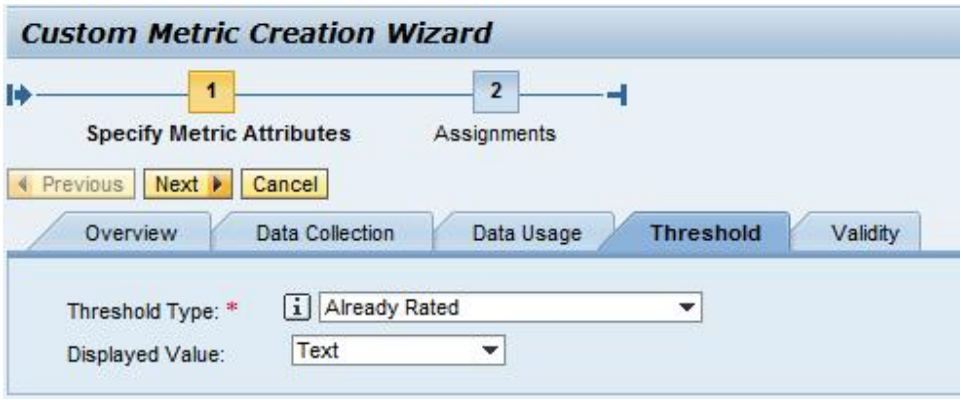


Don’t worry about the parameter PROCNAME, it will be ignored.

On the next tab you select whether you want to collect data for reporting.



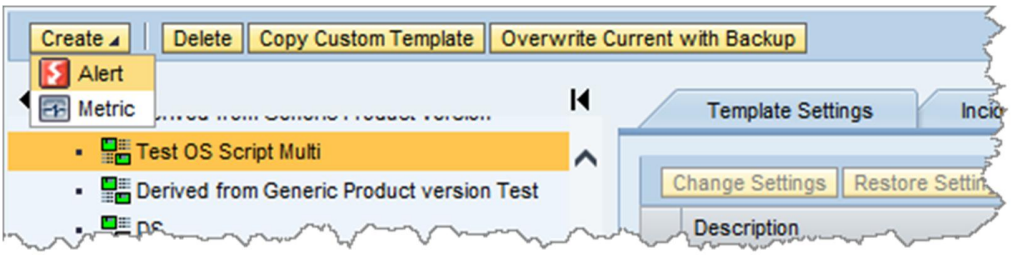
On the tab “Threshold” you have to select “Already rated”. To display the result text select “Text” as display value.



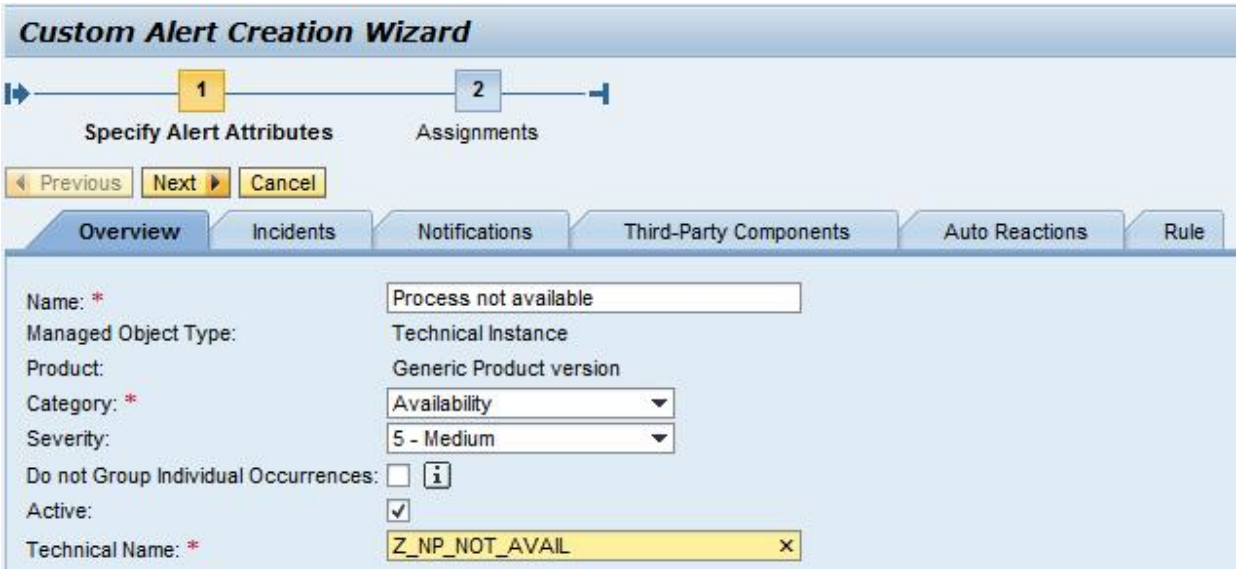
Now you can switch to the next step. We don’t have any alerts yet. So just click “Finish”.

Don’t forget to save your template.

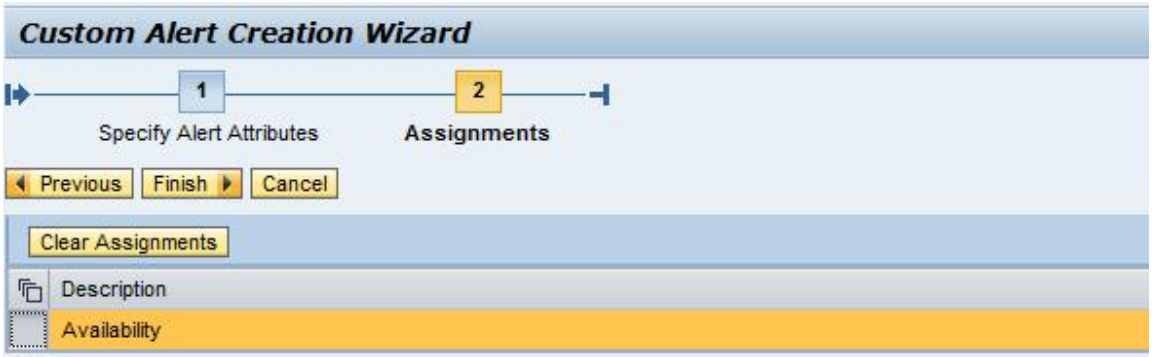
Create an alert for your metric.



Make sure the alert has the same category as the metric, otherwise you cannot assign the metric to the alert.



Select you metric on the next screen and press “Finish”.

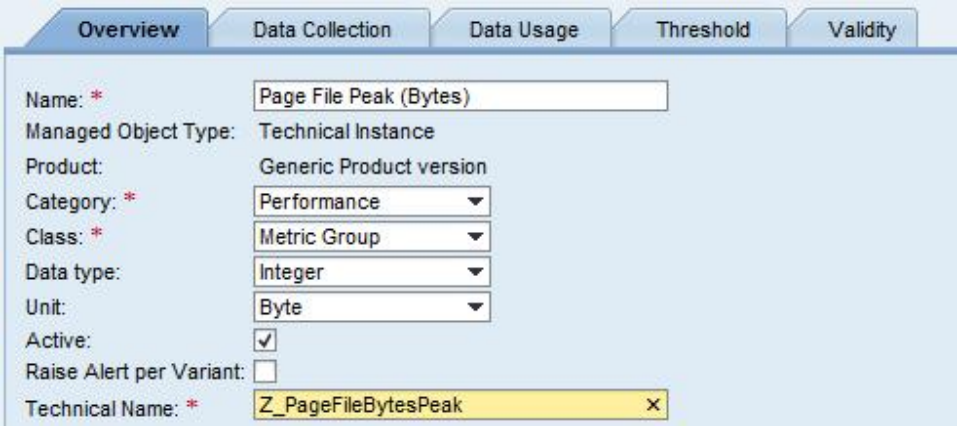


Don’t forget to save you template.

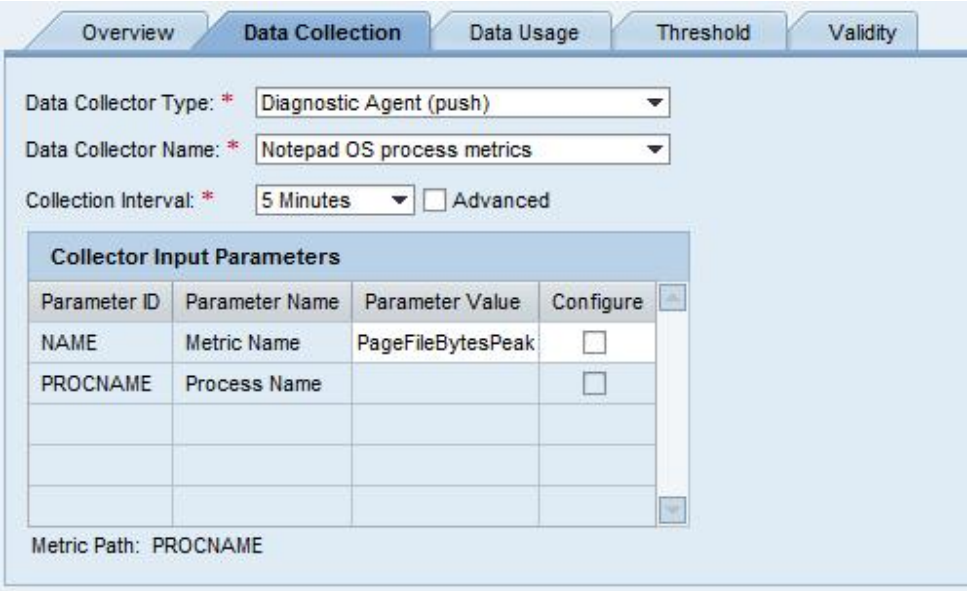
For the second example we take a metric with a *path* part. This metric will return one value for each process that fits the process name (the OPTION parameter in the data collector template).

```
<metric type=""integer"" name=""PageFileBytesPeak"" path=""PROCNAME="$objItem.Name""" value="""$objItem.PageFileBytesPeak"""/>
```

Unlike for the single metric you have now to select “Metric Group”. Also make sure the data type is set to “Integer”.



On the next tab you again select out data collector. Please make sure that the setup is done as below. Metric name should not be set to configure, but be maintained in the upper table.

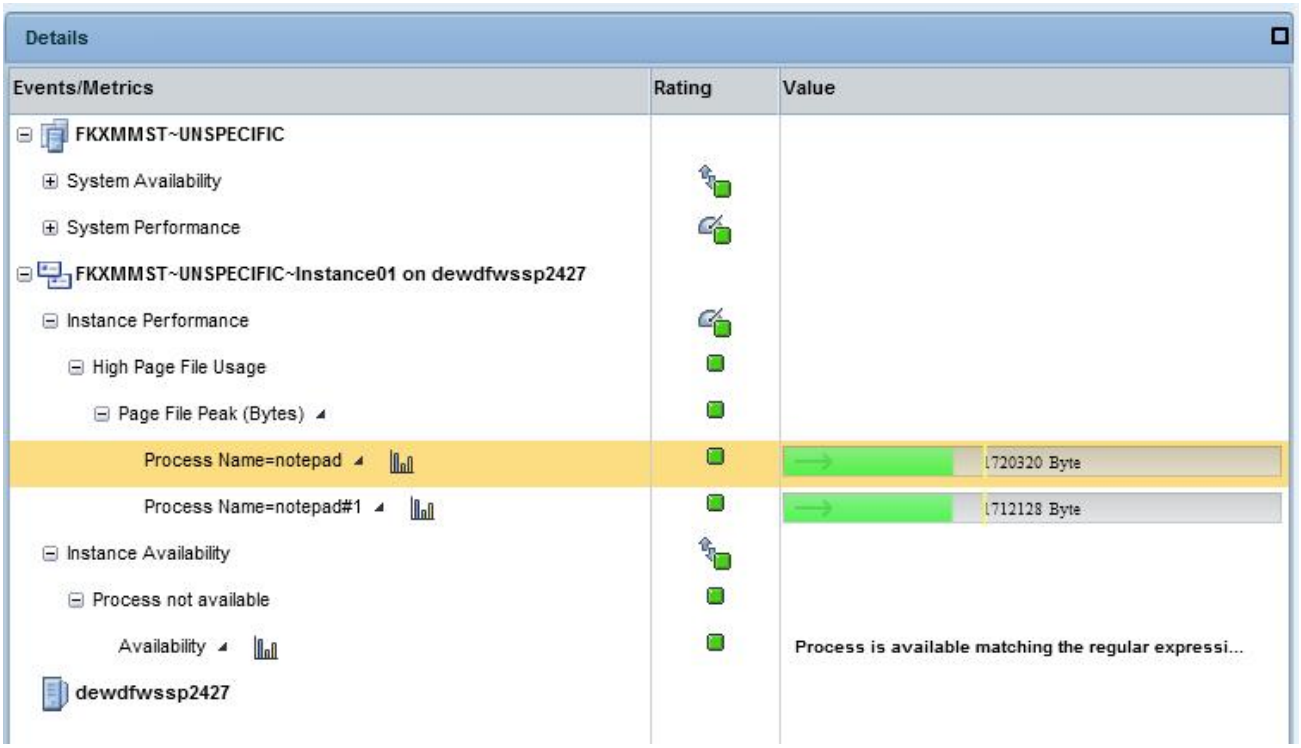


The PROCNAME will automatically be taken into account for the metric path.

The rest of the setup is analog to the fist metric. Create an alert for this metric as well. Then you can go ahead and apply and activate the template to the SAP or non-SAP system you want to monitor.

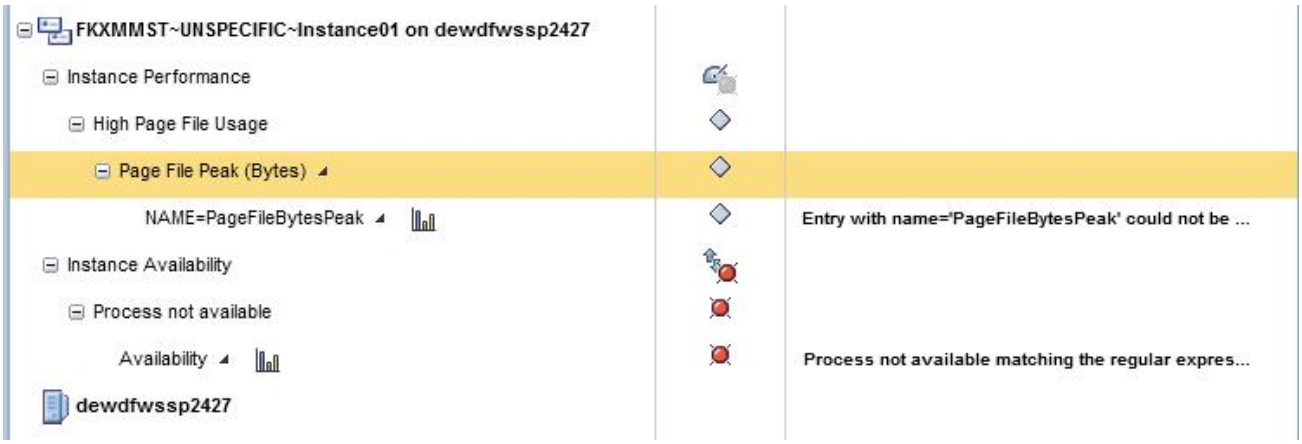
RESULT

After the activation of the monitoring the metrics become visible in the System Monitoring application.



As you can see there is one entry for each returned Notepad process. In our example the metric just extends the process name with an #1, but this depends on the script you write. You sure can do this more sophisticated (e.g. by adding the process ID).

If no process is found the Availability alert will turn red. The other metric will stay grey because nothing that fits is returned in the result set.



APPENDIX

Powershell Script used as Example

```
Param(
[Parameter(Mandatory=$True,Position=1)]
[string]$ProcessName
)

$strComputer = "."
$counter=0

$colItems = get-wmiobject -class "Win32_PerfFormattedData_PerfProc_Process" -namespace "root\cimv2" -
computername $strComputer | Where-Object {$_.Name -match $ProcessName}

# result output
foreach ($objItem in $colItems) {
          write-host -Separator "" "<metric type=""string"" name=""ProcessName""
value="""$objItem.Name"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""CreatingProcessID""
path=""PROCNAME="$objItem.Name""" value="""$objItem.CreatingProcessID"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""ElapsedTime""
path=""PROCNAME="$objItem.Name""" value="""$objItem.ElapsedTime"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""HandleCount""
path=""PROCNAME="$objItem.Name""" value="""$objItem.HandleCount"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IDProcess""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IDProcess"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IODataBytesPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IODataBytesPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IODataOperationsPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IODataOperationsPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IOOtherBytesPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IOOtherBytesPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IOOtherOperationsPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IOOtherOperationsPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IOReadBytesPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IOReadBytesPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IOReadOperationsPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IOReadOperationsPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IOWriteBytesPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IOWriteBytesPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""IOWriteOperationsPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.IOWriteOperationsPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PageFaultsPersec""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PageFaultsPersec"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PageFileBytes""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PageFileBytes"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PageFileBytesPeak""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PageFileBytesPeak"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PercentPrivilegedTime""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PercentPrivilegedTime"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PercentProcessorTime""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PercentProcessorTime"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PercentUserTime""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PercentUserTime"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PoolNonpagedBytes""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PoolNonpagedBytes"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PoolPagedBytes""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PoolPagedBytes"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PriorityBase""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PriorityBase"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""PrivateBytes""
path=""PROCNAME="$objItem.Name""" value="""$objItem.PrivateBytes"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""ThreadCount""
path=""PROCNAME="$objItem.Name""" value="""$objItem.ThreadCount"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""VirtualBytes""
path=""PROCNAME="$objItem.Name""" value="""$objItem.VirtualBytes"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""VirtualBytesPeak""
path=""PROCNAME="$objItem.Name""" value="""$objItem.VirtualBytesPeak"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""WorkingSet""
path=""PROCNAME="$objItem.Name""" value="""$objItem.WorkingSet"""/>"
          write-host -Separator "" "<metric type=""integer"" name=""WorkingSetPeak""
path=""PROCNAME="$objItem.Name""" value="""$objItem.WorkingSetPeak"""/>"

          $counter=$counter+1
}

# already rated metric for process availability
If ($counter -gt 0) {
```

```
        write-host -Separator "" "<metric type=""string"" name=""Availability"" value=""Process is
available matching the regular expression "$ProcessName""" rating=""1""/>"
        }
Else {
        write-host -Separator "" "<metric type=""string"" name=""Availability"" value=""Process not
available matching the regular expression "$ProcessName""" rating=""3""/>"
        }

# number of processes
write-host -Separator "" "<metric type=""integer"" name=""ProcessCount"" value="""$counter"""/>"
```

Batch File used the start Powershell Script

```
@echo off

set processname=%1%

if %1%a==a set processname=saphostexec

Powershell -ExecutionPolicy Bypass -File .\PowershellProcessMonMulti.ps1 %processname%
```