



EMC® Watch4net®

Version 6.3u2

Mapped Property Tag Filter Admin Guide

P/N: pending

Rev : 40030

EMC²

Copyright ©2009 - 2014 EMC Corporation. All rights reserved. Published in the USA.

Published April, 2014

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC², EMC, and the EMC logo are registered trademarks or trademarks of EMC Corporation in the United States and other countries. All other trademarks used herein are the property of their respective owners.

For the most up-to-date regulatory document for your product line, go to EMC Online Support (<https://support.emc.com>).

Contents

Preface	iii
Purpose	iii
Intended Audience	iii
Prerequisites	iii
Document Organization	iv
Common Abbreviations and Acronyms	iv
Where to get help	v
1 Description	1
2 Installation	3
3 Configuration	5
1 Configuration example	5
2 Configuration elements description	6
3 Map File	7
4 Other tools	9

Preface

Purpose

The purpose of this guide is to provide configuration and usage information about the Mapped Property Tag Filter.

Intended Audience

APG administrators and system integrators, as well as users responsible of managing data collection configuration should read this document.

Prerequisites

- The *APG* collector manager must be installed. This filter module may be used with *APG* versions 5.0 and later.
- A map file of keys and values must be specified for the filter to run correctly.

Document Organization

1. Description	A description of this filter function.
2. Installation	Instructions on how to install this filter.
3. Configuration	Instructions on how to configure this filter.
4. Other tools	A description of other tools provided with this filter.

Common Abbreviations and Acronyms

Not Applicable

There are no abbreviations used in this document.

Where to get help

EMC support, product, and licensing information can be obtained as follows:

PRODUCT INFORMATION - For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

<https://support.emc.com>

TECHNICAL SUPPORT - Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

Description

This APG filter is used to tag raw values with properties that are defined in a map file. Tagging will be performed on values identified by their key property. A new property will be added to the raw value. This property will contain the match for the key property found in the raw value. The new property will not be created if this key property does not exist in the raw value or no exact match is defined for its value. There are other mode switches which allow more flexibility when tagging the raw values. These switches will be described in the setup section.

The following is an example of use for the mapped-property tag filter. Let's suppose we will be tagging the raw values that have a property called keyProperty:

Sent raw value's properties:

- RawValue1: name="test1" keyProperty="key1"
- RawValue2: name="test1" keyProperty="key2"

The property map file contains:

```
key1    newPropertyValue1  
key3    newPropertyValue3
```

In this case, the filter's output would be:

- RawValue1: name="test" keyProperty="key1" newProperty="newPropertyValue1"
- RawValue2: name="test1" keyProperty="key2"

Only the raw value RawValue1, named "test1" has been modified. Why?

Its name matches the criteria: its keyProperty value, which is "key1", was found in the property map file, so a new property named newProperty was added to this raw value. RawValue2's keyProperty value was not defined in the map file, therefore no property was generated for this raw value.

Filter limitations

Only exact matches are supported between the raw value's key property value and the key in the map file.

2

Installation

The Mapped Property Tag Filter module may be installed like any other module. The package name is: “mapped-property-tag-filter”. Use the module manager to install this module.

Then, edit the `APG/Collecting/Collector-Manager/Default/conf/collecting.xml` file (location may vary depending upon installation location and Collector-Manager instance name) and insert the following line in the existing configuration. Please refer to the *APG* Backend Administration Guide if unsure on how to do this.

```
<?xml version="1.0" encoding="UTF-8"?>
<config ...>
  <connectors>
    ... connector already defined ...
  </connectors>
  <filters>
    ... next filter ...
    <filter enabled="true" name="MyFilter" next="NextFilter"
      conf="Mapped-Property-Tag-Filter/Default/conf/MyFilterConf.xml" />
    ... previous filter ...
  </filters>
```

The “MyFilterConf.xml” configuration file should be located in the APG/Collecting/Mapped-Property-Tag-Filter/Default/conf/ directory. This file will contain your customized configuration for this filter. Create and edit as explained in this document’s Configuration section. Then restart the *APG* Collector Manager service.

Configuration

This section contains a configuration example as well as the description of the available configuration elements.

1 Configuration example

Let's use the following example in order to create a configuration for this filter:

```
<?xml version="1.0" encoding="UTF-8"?>
<mapped-property-tag-filter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="mapped-property-tag-filter.xsd" tagOnRefreshOnly="false">
  <tagTask removeOldProperty="true">
    <mappedPropertyFileName>conf/example</mappedPropertyFileName>
    <keyPropertyName>StorType</keyPropertyName>
    <fieldSeparator> </fieldSeparator>
    <newPropertyName>parttype</newPropertyName>
  </tagTask>
</mapped-property-tag-filter-config>
```

Note: Please use an XML editor which supports XML schema validation to edit this configuration file.

2 Configuration elements description

Elements and attributes are as follows:

- <mapped-property-tag-filter-config> element contains only one attribute, tagOnRefreshOnly, which should be set to false in most circumstances.
- The <mapped-property-tag-filter-config> element must contain at least one <tagTask> element. The <tagTask> element has only one attribute: removeOldProperty. It can be set to true or false. Setting this to true will remove the key property from the raw value if a match was found and the new property was generated. Several <tagTask> elements may be defined here, to generate new properties from any existing property contained in the raw value handled by this filter. Multiple <tagTask> may use the same keyPropertyName, which will result in several new properties based on the same property value.

Elements nested in a `<tagTask>` element are all mandatory:

- `<mappedPropertyFileName>`: The name of the map file which contains keys and values. In this case, it is set to `conf/example`, which is described below.
- `<keyPropertyName>`: The name of the property that is being checked in the raw value. In this case, `StorType`.
- `<fieldSeparator>`: the character used to separate key and value in the mapped property file. In this case, `TAB` is used, which is a good choice as it should never appear in the key or the value.
- `<newPropertyName>`: the name of the added property.

3 Map File

The map file contains associations between a key value and a mapped value. The separator character between these elements is user definable. Normally, a `TAB` character is a good choice, while `SPACE` is not as it may be used the property value. The syntax of this file is as follows (the comma is used here as the separator character to enhance legibility):

```
key1,value1  
key2,value2  
key3,value3  
....
```


Other tools

A few tools have been provided to better use this filter. These tools can be used to generate the map file used by this filter.

extractSysObjectOID.sh

A typical usage for this filter would be to replace a sysObjectOID (.1.3.6.1.4.1.311.1.1.3.1) by a human readable value (Microsoft Windows NT), so that it may be possible to create APG reports by expanding on this mapped value, instead of the sysObjectOID. In order to do this, a map file which matches the OID to it's human readable description must be generated. This map file can be created and edited by hand but the information it requires can also be extracted from a MIB file. The extractSysObjectOID.sh script, provided with this filter, can be used do this.

These are the steps to follow in order to use this tool:

1. Find the MIB file which contains the required sysObjectOID. Unfortunately, hardware and software vendors do not follow strict standards when defining sysObjectOIDs and it may be difficult to find the MIB which contains them. However, some vendors provide this MIB and name them explicitly (vendorName-products-MIB.mib). In this particular case, the MIB we are looking for is the Microsoft mib (msft.mib).

2. Make sure the MIB contains at least an enterprise node (.1.3.6.4.X). In this case, the enterprise is Microsoft and it's IANA assigned number is 311.
3. By default, the extractSysObjectOID.sh script will extract all leaves that are located under the .1.3.6.1.4.X node. In this case, these leaves are the versions of the Windows operating system, as shown in the MIB file below.

```
MSFT-MIB DEFINITIONS ::= BEGIN

IMPORTS
    enterprises
        FROM RFC1155-SMI;

microsoft      OBJECT IDENTIFIER ::= { enterprises 311 }
software       OBJECT IDENTIFIER ::= { microsoft 1 }
systems        OBJECT IDENTIFIER ::= { software 1 }
os             OBJECT IDENTIFIER ::= { systems 3 }
windowsNT      OBJECT IDENTIFIER ::= { os 1 }
windows        OBJECT IDENTIFIER ::= { os 2 }
workstation    OBJECT IDENTIFIER ::= { windowsNT 1 }
server         OBJECT IDENTIFIER ::= { windowsNT 2 }
dc             OBJECT IDENTIFIER ::= { windowsNT 3 }

END
```

4. Retrieve the MIB information.

Since nothing else is defined in this MIB, we will retrieve all of it's OIDs by simply running the script, without arguments:

```
./extractSysObjectOIDs.sh msft.mib.txt
.1.3.6.1.4.1.311.1.1.3.1.1  microsoft workstation
.1.3.6.1.4.1.311.1.1.3.1.2  microsoft server
.1.3.6.1.4.1.311.1.1.3.1.3  microsoft dc
.1.3.6.1.4.1.311.1.1.3.2    microsoft windows
```

The output can simply be redirected in a map file, or simply redirected in an existing map file.

It is also possible to get some information from one of the leaf parent node. All that's required in this case is to specify the number of levels to go back up from the leaf level. One must also note the two underscore characters provided to the script, which meaning will be explained below.

```
./extractSysObjectOIDs.sh msft.mib.txt _ _ 2
```

```
.1.3.6.1.4.1.311.1.1.3.1.1  microsoft os workstation
.1.3.6.1.4.1.311.1.1.3.1.2  microsoft os server
.1.3.6.1.4.1.311.1.1.3.1.3  microsoft os dc
.1.3.6.1.4.1.311.1.1.3.2    microsoft systems windows
```

In this case, the extra information was retrieved from two hops above the leaf nodes. Specifying only 1 level above the leaf node would have output the following file, which might be more useful, depending upon the intended usage:

```
.1.3.6.1.4.1.311.1.1.3.1.1  microsoft windowsNT workstation
.1.3.6.1.4.1.311.1.1.3.1.2  microsoft windowsNT server
.1.3.6.1.4.1.311.1.1.3.1.3  microsoft windowsNT dc
.1.3.6.1.4.1.311.1.1.3.2    microsoft os windows
```

Other options for this tool is the inclusion of certain OIDs only (both OIDs separated by a comma, and every leaf under them):

```
./extractSysObjectOIDs.sh msft.mib.txt .1.3.6.1.4.1.311.1.1.3.1.1,.1.3.6.1.4.1.311.1.1.3.1.2
.1.3.6.1.4.1.311.1.1.3.1.1  microsoft workstation
.1.3.6.1.4.1.311.1.1.3.1.2  microsoft server
```

And the exclusion of certain OIDs (the first OID is included, the second one is excluded, as well as every leaf under it)

```
./extractSysObjectOIDs.sh msft.mib.txt .1.3.6.1.4.1.311.1.1.3.1 .1.3.6.1.4.1.311.1.1.3.1.2
.1.3.6.1.4.1.311.1.1.3.1.1  microsoft workstation
.1.3.6.1.4.1.311.1.1.3.1.3  microsoft dc
```

Or all OIDs, except one:

```
./extractSysObjectOIDs.sh msft.mib.txt _ .1.3.6.1.4.1.311.1.1.3.1.2
.1.3.6.1.4.1.311.1.1.3.1.1  microsoft workstation
.1.3.6.1.4.1.311.1.1.3.1.3  microsoft dc
.1.3.6.1.4.1.311.1.1.3.2    microsoft windows
```

createOid2TypeMap.sh

It is possible to extract a lot of information out of MIB files by using this tool. The generated output can be directly used as a map file, with a TAB separator character. createOid2TypeMap.sh

If you are already using EMC SMARTS and have access to its oid2type files, it is possible to retrieve the content of those files and output a map file by using the createOid2TypeMap.sh script.

Its usage is very simple:

createOid2TypeMap.sh someVendorOid2type.conf output

The output file will contain the map made of:

The sysObjectOID, separated by a TAB character, followed by the VENDOR and MODEL values.

```
.1.3.6.1.4.1.43.1.8.29 3Com SuperStackII Desktop Switch SW Version:2.10  
.1.3.6.1.4.1.43.10.27.4.1.2.1 3Com 3Com SuperStackII  
...
```