



EMC® Watch4net®

Version 6.3u2

Group Filter Administration Guide

P/N: pending

Rev

EMC²

Copyright ©2013 - 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
Intended Audience	iii
Prerequisites	iii
Document Organization	iii
Where to get help	v
1 Product Overview	1
2 Architectural Overview	3
1 Group Member	3
2 Group	3
3 Connection Source	3
4 Connection Destination	3
3 Things to do before installation	5
4 Installation	7
1 APG 4.2 and over	7
5 Configuration	9
1 Collector Manager configuration	9
2 <i>Group Filter</i> configuration	10
2.1 Configuration Overview	10
2.2 Global configuration	10
2.3 Group Members	11
6 Things to do after configuration	17
7 Appendix A	19
1 Configuration Example	19

Preface

The purpose of this guide is to provide information about the *Group Filter* for APG. It contains all the necessary information to install and configure the filter.

Intended Audience

This guide is intended to be read by any *APG* administrator who will configure the *Group Filter* to create groups of related values and tag values with group properties.

Prerequisites

It is assumed that the *APG* application has been properly installed and configured on the server.

Document Organization

1. Product Overview	This section gives an overview of the <i>Group Filter</i> .
5. Configuration	This section explains how to configure the filter.

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.

Product Overview

The *Group Filter* for APG provides a solution for cases where properties of metric values belonging to different devices or parts need to be shared among the various metric values.

In doing this, it creates Groups of related RawValues. Groups can contain RawValues for different component types. The configuration for the filter defines connections between values by specifying properties in one component type which must match values in another component type.

In addition, the filter provides for the storage and retrieval of Group Properties. These are properties that are meant to be shared among members of the group. The configuration specifies that certain properties for values of certain type are stored as Group Properties. The configuration also specifies that these properties can be pulled from the Group Property storage and added to other values.

For example, in the storage domain, a MetaLUN or MetaHead consists of one or more component LUNs called meta members. The MetaHead inherits the same characteristics as its members. Examples of characteristics would be disk technology, RAID protection, fast cache enabled, etc.

During discovery, the RawValues for the MetaHead objects do not contain these characteristics while the RawValues for the meta members do.

When configured properly, the Group Filter resolves this problem by creating Groups of related RawValues. Each Group consists of a MetaHead and all of its meta members. The configuration specifies that the meta members store disk information as Group Properties while the MetaHead pulls

the disk information from the Group Properties. In this way, the MetHead RawValues have the disk properties available for display in reports.

This configuration is described in the examples provided in the [5](#) section of this documentation.

2

Architectural Overview

- 1 Group Member
- 2 Group
- 3 Connection Source
- 4 Connection Destination

3

Things to do before installation

4

Installation

1 APG 4.2 and over

An up-to-date version of the *Group Filter* is already provided with your APG distribution. Skip the installation step and go directly to configuration (section [5](#))

5

Configuration

This section describes the different configuration files for the filter.

1 Collector Manager configuration

This filter must be declared in the Collector-Manager <APG-location>/Collecting/Collector-Manager/Default/conf/ file in order to be used.

Add the following section if it is not present, and make sure that the *enabled* attribute is set to *true*:

```
<filter enabled="true" name="VNX-GF1" next="VNX-CRF3" config="Group-Filter/emc-vnx/conf/group-  
filter-vnxblock-meta.xml" />
```

Multiple instances of the *Group Filter* may be configured but each must have its own *filter* tag and configuration file.

2 Group Filter configuration

2.1 Configuration Overview

A configuration file, provided in the *config* attribute of the *filter* tag as described in [Collector Manager configuration](#), must be created for each *Group Filter* instance. This defines global attributes for the filter in general. It also defines each of the group members including

- Conditions — property values or filters which define which values belong to a group member.
- Identity properties — which properties are used to distinguish one value of a group member from another (these are not necessarily the *device* and *part*).
- Connections between values of different type which show how the different values relate to each other.
- Group properties — properties that are shared among members of the group which can be used to tag values.

2.2 Global configuration

The following properties apply to the *Group Filter* as a whole.

data-accessor Defines how the *Group Filter* data will be stored. At present, the only implementation is *MemoryDataAccessor* which keeps everything in RAM.

```
<configuration>
...
  <data-accessor>com.watch4net.apg.v2.collector.plugins.groupfilter.data.mem.
MemoryDataAccessor</data-accessor>
```

property-timeout Defines the timeout for old property values. Property values not updated during this interval, given in minutes, will be removed from the *Group Filter* instance. The default value is 30 minutes.

In the following example, the property timeout is defined as 60 minutes.

```
<configuration>
...
  <property-timeout>60</property-timeout>
```

deleted-metric-frequency Defines how often the filter should check for deleted metric values. The default value is 60 minutes.

In the following example, the deleted metric value frequency is defined as 30 minutes.

```
<configuration>
...
<deleted-metric-frequency>30</property-timeout>
```

deleted-metric-timeout Defines the timeout for old metric values. Metric values not updated during this interval, given in minutes, will be removed from the *Group Filter* instance. The default value is 60 minutes.

In the following example, the deleted metric timeout is defined as 30 minutes.

```
<configuration>
...
<deleted-metric-frequency>30</property-timeout>
```

2.3 Group Members

The following describes the definition of Group Members. A Group Member represents a class of metric values that share a common set of properties. For example, one Group Member might represent disks and another arrays, Group Members can also be more granular and represent only disks of a certain technology or size or any combination of available properties.

condition Defines a condition which a metric value must satisfy in order to be considered a member of the class represented by the Group Member. A condition can be either a property/value pair or an APG filter of arbitrary complexity. A Group Member definition can contain multiple conditions. The following two examples have identical results which limit membership in the Group Member class to array (*devtype == Array*) LUN (*parttype == LUN*) metric values which are meta members (*ismetam == 1*).

When creating a filter, note the use of *&*; to replace *&* as is required when creating XML files. It will also be necessary to use *<* in place of *<*.

```
<configuration>
...
<group-member>
...
<condition type="filter">devtype=='Array'&parttype=='LUN'&ismetam=='1'</condition>
```

```
<configuration>
...
<group-member>
...
<condition property="devtype">Array</condition>
<condition property="parttype">LUN</condition>
<condition property="ismetam">1</condition>
```

identity The identity defines which properties can be used to distinguish one object of a Group Member from another. Often these properties may be *device* and *part* but they do not have to be.

For example, in the *metaLUN* example described above ([Product Overview](#)), all the meta members in a particular Group instance must have a *metahid* which matches the meta head *partid*. Thus, for meta head, the identity is

```
<configuration>
...
<group-member>
...
<identity>device</identity>
<identity>partid</identity>
```

and, for meta member, the identity is

```
<configuration>
...
<group-member>
...
<identity>device</identity>
<identity>metahid</identity>
```

With this definition of identity for meta member, we will find that many distinct meta member instances (*i.e.* LUNs with different *parts*) will have the same identity. This is so because, for the purpose of this specific use case, we are only interested in the meta members that go with a given meta head. Thus, even though there will be many distinct meta members associated with a meta head, they are all the same **for the purpose of this use case** and the identity definition reflects this.

destination As described in [Connection Destination](#), a Destination is one side of the Connection between two Group Members. Every Connection must have exactly one Destination and one

Source side which use the same value for the *name* property. The *destination* in a Group Member configuration just specifies which Group Member is the destination side of the connection between two Group Member instances.

In the example below for the metaLUN use case, the configuration says that, for the "metahead" connection, the Destination properties that must match a Source object are *device* and *metahid* because, as shown in the example above, the identity for the meta members consist of these two properties.

```
<configuration>
...
  <group-member>
    ...
    <destination name="metahead"/>
```

source As described in [Connection Source](#), a Source is one side of the Connection between two Group Members. Every Connection must have exactly one Destination and one Source side which use the same value for the *name* property. Configuration of the Source side of the Connection also requires indicating which properties of the Source Group Member are associated with the identity properties of the Destination side of the connection.

By default, the properties on the Source side are the same as the identity properties so, in the example below, the connection properties for the "metahead" connection are *deviceid* and *partid*. The properties are matched in order so, taken together with the *destination* example above, meta head and meta members are in the same Group instance when the meta head *device* property matches the meta member *device* property and the meta head *partid* property matches the meta member *metahid* property.

```
<configuration>
...
  <group-member>
    ...
    <source name="metahead"/>
```

It may also be necessary to specify the Source properties explicitly. This can be done by defining children of the *source* tag. Children can be specified using *<key>* or *<list>*. A *<key>* child's value is the name of a property to use when making a connection. The above example could therefore be defined using

```
<configuration>
...
```

```
<group-member>
...
<source name="metahead">
  <key>device</key>
  <key>partid</key>
</source>
```

A *<list>* child's value is also the name of a property to use when making a connection. In this case, however, the property value is expected to contain a list each of whose items is a value to match against the corresponding destination side property.

For example, making a small adjustment to the metaLUN use case we have been using for the examples, we suppose a new property *partidlist* and define the Source side of the connection as

```
<configuration>
...
<group-member>
...
  <source name="metahead">
    <key>device</key>
    <list separator="|" ">partidlist</list>
  </source>
```

In this case, *partidlist* is expected to contain values which are lists of *metahid* names separated by a '|' character. The default separator is ',' but can, as we have seen, be overridden by any string.

Given this definition for Source, and supposing a meta head has a *partidlist* property with a value of "123|255|389", then it describes a Group instance containing this meta head as well as the meta member instances with matching *device* as well as *metahid* values of "123", "255", and "389".

addtogroup The *addtogroup* configuration item indicates which properties from metric values that are members of the Group Member's class are to be added to the Group's properties. These Group properties will be available for tagging of any metric values during processing.

For example, given the following configuration, which is taken from the meta member Group Member being developed in these examples, whenever a meta member metric value is processed, its values for the *disktype*, *disksize*, and *isfcache* properties are added to the Group instance for that meta member.

```
<configuration>
...
<group-member>
...
<addtogroup>disktype</addtogroup>
<addtogroup>disksize</addtogroup>
<addtogroup>isfcache</addtogroup>
```

pullfromgroup The ultimate purpose of the *Group Filter* is to tag metric values with properties taken from related metric values. The *pullfromgroup* item indicates which property values should be copied from the Group instance to metric values which are members of the class described by the associated Group Member.

For example, given the following configuration, which is taken from the meta head Group Member being developed in these examples, whenever a meta head metric value is processed, it will be tagged with the Group instance's values of the *disktype*, *disksize*, and *isfcache* properties (assuming a meta member metric value has been processed to add them to the Group instance).

```
<configuration>
...
<group-member>
...
<pullfromgroup>disktype</pullfromgroup>
<pullfromgroup>disksize</pullfromgroup>
<pullfromgroup>isfcache</pullfromgroup>
```

As noted, it is possible that a property value has not yet been added to the Group instance. The *default* property to *pullfromgroup* provides a value to be use if no actual values have been stored. In the example below, the string "N/A" will be used as the value of *disktype* if no values have yet been store in the Group instance.

```
<configuration>
...
<group-member>
...
<pullfromgroup default="N/A">disktype</pullfromgroup>
```

Given that there can be multiple metric values as members of the class described by a Group Member, it is possible that different values of a given property may be stored in a Group instance. The following optional properties are available to control how this is handled.

- separator
- maxitems
- sort
- substitute

By default, if there are multiple property values stored, the *pullfromgroup* behavior is to create a list of all the items stored.

The default separator for the created list is '|' but this may be overridden by the *separator* property.

By default, all the items are output but this may be overridden by the *maxitems* property which specifies how many to output. If this property is specified, and there are more than the specified value, the output will include ellipsis (...) to indicate that there are missing items in the list.

By default, the list items are retrieved in an indeterminate order but this may be overridden using the *sort* property which specifies that the list items should be sorted in lexicographical order (A B C a b c). Note that this can add significant overhead to processing so it should be used sparingly.

As described above, the default behavior when there are multiple property values is to output a list of items which can be controlled in various ways. This can be overridden by the *substitute* property which provides a string value to use in place of the list if there are multiple values. If there is a single value and the substitute property is included, the output consists of the actual value and not the substitute string. The *substitute* property overrides the others so, if *substitute* is specified along with list-related properties, the substitute value will be output.

As an example, if we have a metric value property *val* with the values "c", "a", "B", "C", "b", "A" for a particular Group instance, then the following table describes the behavior given various *pullfromgroup* property definitions.

	c a B C b A
separator="@. "	c@. a@. B@. C@. b@. A
maxitems="3"	c a B ...
sort="yes"	A B C a b c
substitute="MixedData"	MixedData
substitute="MixedData" sort="yes"	MixedData
sort="yes" maxitems="2"	A B ...
sort="yes" maxitems="6"	A B C a b c

6

Things to do after configuration

When the collector is installed and configured, APG is ready to collect new data from the XML source. The following steps can be taken:

1. Start the APG collecting service.
2. Look at the collecting logs to make sure that everything is starting up as it should.

Appendix A

1 Configuration Example

A completed configuration based on the discussions above is:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.watch4net.com/GroupFilter">

  <data-accessor>com.watch4net.apg.v2.collector.plugins.groupfilter.data.mem.
MemoryDataAccessor</data-accessor>

  <!-- Meta head gets disksize and disktype of its members if they are the same for all
members.
  It they differ, it is set to Mixed. Members are mapped to its meta head via metahid that
matches
  the partid of the meta head.
  isfcache is set to enabled if it is enabled for at least one meta member. -->
```

```
<!-- Metrics of meta members -->
<group-member>
  <condition property="devtype">Array</condition>
  <condition property="parttype">LUN</condition>
  <condition property="ismetam">1</condition>
  <identity>device</identity>
  <identity>metahid</identity>
  <destination name="metahead"/>
  <addtogroup>disktype</addtogroup>
  <addtogroup>disksize</addtogroup>
</group-member>

<!-- Metrics of meta head-->
<group-member>
  <condition property="devtype">Array</condition>
  <condition property="parttype">LUN</condition>
  <condition property="ismetah">1</condition>
  <identity>device</identity>
  <identity>partid</identity>
  <source name="metahead"/>
  <pullfromgroup substitute="MixedData">disktype</pullfromgroup>
  <pullfromgroup substitute="MixedData">disksize</pullfromgroup>
</group-member>

</configuration>
```