



EMC® Watch4net®

Version 6.3u2

Event Aggregator Administration Guide

P/N: pending

Rev : 45433

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
Where to get help	v
1 Product Overview	1
1 Configuration Overview	1
2 Architecture	3
1 Initial Processing Elements	5
2 Aggregator	5
3 Events Formatter	5
4 Raw Values Formatter	6
3 Configuration	7
1 Event Processing Manager Configuration	7
2 Event Aggregator Configuration	8
3 Socket Configuration	12
4 Aggregation Components	15
1 Value renderer	15
1.1 AverageRenderer	16
1.2 CountRenderer	16
1.3 LastRenderer	16
1.4 MaxRenderer	16
1.5 MinRenderer	16
1.6 SumRenderer	16
2 Mapping engine	17

- 2.1 DefaultMappingEngine 17
- 3 Mapper 18
 - 3.1 BitMapper 18
 - 3.2 IntMapper 18
 - 3.3 IPFilterMapper 19
 - 3.4 MaskMapper 20
 - 3.5 StringMapper 20
 - 3.6 RegexMapper 21
- 5 Configuration Examples 23
 - 1 Netflow Time Series Output 23
 - 2 Broadsoft Event Output 25

Preface

Purpose

This guide describes the common usage of the Event Aggregator, which includes all the necessary information to install and configure the module.

Intended Audience

This guide is intended to be read by any *APG* administrator who will configure this module to convert events into a set of fewer events or into a time series of raw values.

Prerequisites

The Event Aggregator can only be used within an instance of the Event Processing Manager; version 5.0 and following are supported.

Document Organization

1. Product Overview	This section provides a quick overview of this product, as well as configuration overview steps.
2. Architecture	This section describes the global architecture of this product.
3. Configuration	This section describes the core configuration of the Event Processing Manager and Event Aggregator.
4. Aggregation Components	This section describes the different value renderer and property mappers available in the configuration used to determine the property set and type of aggregation applied to incoming events.
5. Configuration Examples	This section contains complete configuration examples for both time series raw value, and event output.

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 Event Aggregator allows *APG* to convert large event based data into fewer events or a time series of raw values.

This is achieved through aggregations which greatly improve the performance of report generation. The Event Aggregator depends on modules that output events further-up in the event processing chain which provide the data to aggregate.

1 Configuration Overview

Follow the workflow steps below to configure the Event Aggregator and its complement modules:

1. Refer to the Event Aggregator architecture section to gain an understanding of the module. See [Architecture](#) on page 3
2. Configure other processing elements preceding the Event Aggregator in the flow. See [Initial Processing Elements](#) on page 5.
3. Configure the **processing.xml** file for the Event Processing Manager. See [Event Processing Manager Configuration](#) on page 7.

4. Configure one or more Event Aggregator configuration files depending on your number of sources. See [Event Aggregator Configuration](#) on page 8. Also refer to [Aggregation Components](#) on page 15 to assist you in your configurations.
5. For time series output, configure the Cache connector where raw times series values will be sent, in the **timeseries-connector.xml** file. See [Socket Configuration](#) on page 12.
6. Configure other elements in the event flow for events output, or in the raw value flow for time series output. See corresponding module documentation.
7. Restart the Event Manager.
8. For time series output, make sure the Backend is running. Make sure any other modules and/or services are restarted if you made any changes to accommodate the Event Aggregator output.

Architecture

The Event Aggregator depends on modules further up in the event processing chain to provide it with event data; these are referred to as the initial processing elements as they take place earlier in the flow than the Event Aggregator. The Event Aggregator is then configured to retrieve this event data and output it either to an Events formatter, where the output is a set of events aggregated from the input, or to a Raw values formatter, where the output is a time series of raw values similarly aggregated from the input. Figure 1 on page 4 describes the way the data flow is handled and transformed into fewer events or time series values.

In the case of event output, events are sent to other elements in the event processing chain; in the case of time series output, raw values are sent to other elements in *APG*'s standard raw value processing chain using a Cache connector.

The Event Aggregator uses mappers and a value renderer (explained in [Aggregation Components](#) on page 15) to define the properties that will pertain to each outputted event or raw value, and how they will be aggregated.

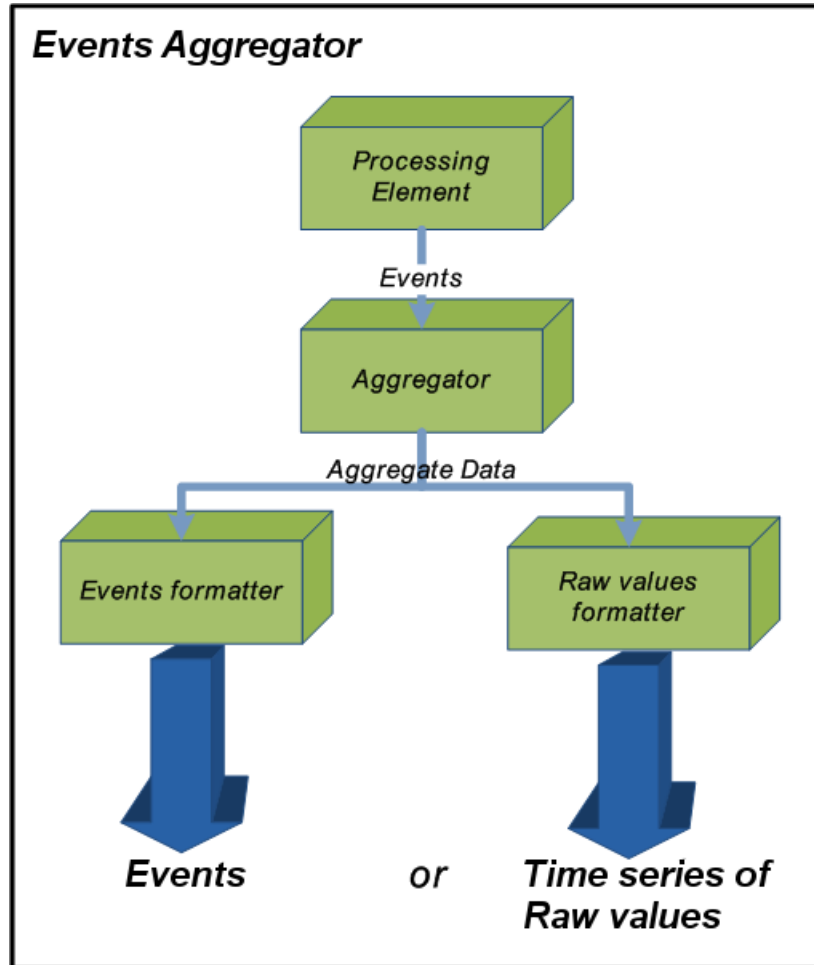


Figure 1: Event Aggregator Architecture

The following is a description of the important elements involved when using the Event Aggregator.

1 Initial Processing Elements

Other modules in the event processing chain, for example the Flow Listener, or the Broadsoft VoIP Collector, feed the Event Aggregator with raw event base data.

Events are sent to the Event Aggregator as soon as they are fetched and no history is kept about them in these previous modules. Additionally, only a select set of properties will be maintained for each event (as configured via mappers, see [Mapping engine on 17](#)) with the rest being discarded. For this reason you might want to, for example, configure an additional event flow that bypasses the Event Aggregator and is written directly to the Datastore, in this case maintaining all event properties.

There could be other modules in the events processing chain before the events reach the Event Aggregator, for example the Event Property Tagger, which can add properties to an event based on properties and their values it already has. For more information, see the *Event Property Tagger User Guide*.

2 Aggregator

This is where the fundamental activity of the Event Aggregator takes place, where raw event based data is transformed into aggregated data. The aggregator is configured to use property mappers and a mapping engine to format a select set of the incoming event's properties and create aggregates of these events which all have matching properties and property values. The results of the mapping are sets of events with matching properties over every aggregation period, each of which are then aggregated into a single event or raw value by a value-renderer, which performs aggregations such as averages or sums on a specified aggregation value.

The timestamp of each output event/raw value corresponds to the last event registered during the aggregation period. The set of properties of each output event/raw value are those configured and matched for by the mappers.

The Aggregator will be configured to output either events or a time series of raw values.

3 Events Formatter

This element takes all the aggregated data generated by the aggregator for each aggregation period to recreate events and then send them to the next processing elements. The formatter will create one event per aggregated key containing every value defined. Beside the fields defined by the configuration, each generated event will have the following fields added:

- `eventTmst`: The timestamps of the last received event for each aggregation.
- `NAME_OF_VALUE`: One field per defined value
- `unitPLUS_NAME_OF_VALUE`: The unit of the value for each aggregation.
- `eventId`: The unique identifier for each aggregation.

4 Raw Values Formatter

This element takes all the aggregated data generated by the aggregator for each aggregation period to format them into APG raw values and then sends them using a Cache connector to the collector Backend directly or via a Socket collector.

Beside the fields defined by the configuration, each generated raw value will have the following fields added:

- `name`: The name of the value for each aggregation.
- `unit`: The unit of the value for each aggregation.

Configuration

To configure the Event Aggregator, each event source must be declared in the Event Processing Manager and have its own configuration file adjusted. As well for time series output, the socket where raw values are sent must be configured.

1 Event Processing Manager Configuration

To use the Event Aggregator you must modify the **processing.xml** file to include a line for each event source you want to aggregate into fewer events or a time series.

The path to the file is:

```
<APG>/Event-Processing/Event-Processing-Manager/<instance name>/conf/  
processing.xml
```

The following is an example for a Netflow source outputting a time series:

```
<!-- Event Aggregator (if aggregates are needed) -->  
<processing-element enabled="true" name="Events-Aggregator" config="Event-Aggregator/Default/  
conf/flow-aggregator-ts.xml" />
```

The following is an example for a Broadsoft source outputting events:

```
<!-- Event Aggregator (if aggregates are needed) -->
<processing-element enabled="true" name="Events-Aggregator" data="otherProcessingElement"
config="Event-Aggregator/Default/conf/voip-aggregator.xml" />
```

Note: The data attribute is added in the Broadsoft example to indicate that the resultant events will continue in the event flow usually reaching the Datastore for storage. It must be present for event output. When outputting a time series, do not include the data attribute since no other event processing elements can work with the time series. The time series will not be processed if you do so.

2 Event Aggregator Configuration

The configuration files for the Event Aggregator are based on the schema: `eventsAggregatorConfiguration.xsd`.

By default, there are three examples of configuration files located at:
`<APG>/Event-Processing/Event-Aggregator/<instance name>/conf`

- The `flow-aggregator-ts.xml` file contains a sample configuration for outputting raw values from a Netflow source. This is the file used for the example below.
- The `flow-aggregator-events.xml` file contains a sample configuration for outputting events from a Netflow source.
- The `voip-aggregator.xml` file contains a sample configuration for outputting events from a Broadsoft source.

These are the files that must be declared for each event source in the **processing.xml** file; see *Event Processing Manager Configuration* above.

You can create another aggregator file for a different source, based on one of these files, name it appropriately, and declare it in the **processing.xml** file.

```
<aggregation-thread>2</aggregation-thread>
<aggregation-period>240</aggregation-period>
<aggregation-trigger>
  <changed />
```

```

<timeout time="6h" />
<aggregation-value-list operator="lower-or-equal" name="min">12</aggregation-value-list>
<aggregation-value-list operator="equal" name="last">69</aggregation-value-list>
<aggregation-value-list operator="greater" name="ifOut">10000</aggregation-value-list>
</aggregation-trigger>
<reuse-aggregates>true</reuse-aggregates>
<force-aggregates-order>false</force-aggregates-order>
<timestamp-field format="sec">FLOW_TMST</timestamp-field>
<destination-config output-type="raw-values">
  <params-list key="group">group</params-list>
  <params-list key="source">Netflow</params-list>
  <params-list key="config-file">conf/timeseries-connector.xml</params-list>
</destination-config>

<aggregation-parameter-list aggregator-name="Source_sum">
  <aggregation-value-list name="ifOut" unit="byte" type="SumRenderer" content="integer"
output="rate">BYTES</aggregation-value-list>
  <aggregation-value-list name="min" unit="byte" type="MinRenderer" content="integer" output=
"rate">FIELD1</aggregation-value-list>
  <aggregation-value-list name="last" unit="byte" type="LastRenderer" content="integer"
output="rate">FIELD2</aggregation-value-list>
  <output-selector>
    <topn-by-key size="100" value="BYTES">
      <key-list>device</key-list>
      <key-list>ip</key-list>
    </topn-by-key>
  </output-selector>
</mapping-engine-config>
  <mapping-engine>DefaultMappingEngine</mapping-engine>
  <mapper-config-list>
    <mapper>IPFilterMapper</mapper>
    <mapping-key-list property-name="device">ROUTER_SRC</mapping-key-list>
  </mapper-config-list>
  <mapper-config-list>
    <mapper>IPFilterMapper</mapper>
    <mapping-key-list property-name="ip">IPV4_SRC_ADDR</mapping-key-list>
    <param-list key="grouped">PublicNet</param-list>
    <param-list key="detailed">10.0.0.0/8</param-list>
    <param-list key="detailed">192.168.0.0/16</param-list>
  </mapper-config-list>
</mapping-engine-config>
<constant-property-list>

```

```

<mapper>IntMapper</mapper>
<mapping-key-list property-name="devid">IF_ID</mapping-key-list>
<mapping-key-list property-name="part">INPUT_SNMP</mapping-key-list>
</constant-property-list>
<constant-property-list>
  <mapper>StringMapper</mapper>
  <mapping-key-list property-name="devdesc">IF_DESC</mapping-key-list>
</constant-property-list>
<hardcoded-property-list property-name="parttype">Interface</hardcoded-property-list>
</aggregation-parameter-list>

```

- **aggregation-thread**: The number of threads that will do the aggregations in parallel.
- **aggregation-period**: The interval between two aggregates trigger ready check.
- **aggregation-trigger**: Defines the conditions to release the aggregates. This section is optional and will behave as having only the *changed* trigger configured. All the available triggers are not mandatory and the aggregates will be release as soon as one trigger is met. When listing multiple triggers, it a good idea to put them in order of the most likely to have its condition met to the less likely to in order to have the best performances.
 - **changed**: Will release the aggregate if at least on value has changed in the time frame (default behavior)
 - **timeout** with time="*time expression*": Will release the aggregate if no values has been updated for a configurable period of time
 - **aggregation-value-list** with name="*value name*" and operator="*equal, not-equal, lower, lower-or-equal, greater or greater-or-equal*": Will compare a given value with a given threshold for a given operator to see if the aggregate should be released.
- **reuse-aggregates**: Optional flag telling if the aggregates can be reused (default behavior). Set this to false when each aggregate is unique in any given amount of time.
- **force-aggregates-order**: Optional flag telling if the incoming aggregates must be kept in order of creation. Use this feature only if the different aggregates are dependent on each other (Default is false).
- **timestamp-field**: The field containing the timestamp in the event. The timestamps in the field is expected to be in seconds (format is optional in this case since the default value is sec). The timestamp of the last event in the aggregation period is kept.

- **destination-config**: The configuration of the destination type with output-type=*“events or raw-values”*. Any parameters for event output will result in static properties in each event. For raw value output, the following parameters are expected:
 - **group**: Storage retention group
 - **source**: Source property
 - **config-file**: The location of the configuration file for the Cache connector where time series will be sent. See [Socket Configuration](#) on page 12.
 - **refresh-on-new-aggregate**: (Optional, will be set at true when omitted) Will determine if the +r action will be set on new aggregates that are released. It's recommended to set this parameter to false when using **reuse-aggregates**=*“false”*.
- **aggregation-parameter-list**, with aggregator-name=*“unique aggregator name”*: The container for particular aggregation settings. Unbounded instance size: You may have different aggregation settings, for example, for the source and destination when using Netflow .
 - **aggregation-value-list** with name=*“value name”*, unit=*“unit”*, type=*“value renderer”*, content=*“decimal or integer”* and output=*“delta or rate”*: The value of this tag must be the exact name of an existing field in the raw data that represents a desired value. The *value name* will be the name of the resulting APG value. The *unit* is simply the unit of the value. The [Value renderer](#) (see page 15) is the class name of the value renderer present in the package `com.watch4net.events.aggregator.rendering`. The *content* specify what kind of variable will be extracted and computed. If omitted, the value will be decimal and it will be fine most of the time. The integer value should be used when dealing with really large numbers. The *output* specify what kind of value will be computed. By default, the value will be a delta but it can be changed to be a rate over the aggregation period in seconds.
 - **output-selector**: Defines the how to select the desired aggregates from all the outgoing aggregator results. This section is optional and will behave as having the *all* option configured. The available selectors are:
 - * **all**: Will select every aggregates (default behavior)
 - * **topn-by-key** with size=*“N value”*, value=*“Sorting value”*: Will select the TopN values grouped by a subset of mapping keys. Each underlying *key-list* tag must be one of the resulting mapping key property and will define a separate topN for each grouped keys value.
 - **mapping-engine-config**: The container of the mapping engine used to create a unique aggregated data key for the current aggregation.

- * **mapping-engine**: The class name of the [Mapping engine](#) (see page 17) present in the package `com.watch4net.events.aggregator.mapping`. Currently, the only engine available is the `DefaultMappingEngine`.
- * **mapper-config-list**: The container of the configuration for each mapper that is used. Unbounded instances size: as many mappers as necessary can be used.
 - **mapper**: The simple class name of the [Mapper](#) (see page 18) present in the package `com.watch4net.events.aggregator.mapping`
 - **mapping-key-list** with `key="mapping key if needed"`, `property-name="property name"`: The value of this tag must be the exact name of an existing field in the raw data and must be present in each raw data instance otherwise the data will be flushed. The *mapping key* represents the logical properties for a subset of values (ex: The source port and source address of a flow to extract the bytes sent by a host). The *property name* will be the name of the APG property that will contain the value of the raw data field.
 - **param-list** with `key="parameter name"`: Some mappers require or can be given a set of parameters with different values.
- **constant-property-list**: This optional list contains standard mapper configuration to extract properties that are believed to be constant and will be extracted the first time a new key is discovered. These properties will be part of the final output value.
- **hardcoded-property-list**: This optional list contains hardcoded properties that will be part of the final output value.

3 Socket Configuration

When outputting time series raw values from the Event Aggregator, you must configure a Cache connector to point to a socket where the output as times series will be sent, either directly to the Backend, or by entering the standard APG raw values flow via a Socket collector.

You do this in the **timeseries-connector.xml** file located at: `<APG>/Event-Processing/Event-Aggregator/<instance name>/conf`

Note: There can be only one destination per source aggregation file (e.g. **flow-aggregator-ts.xml**, ...). If you want to add an additional destination create an additional file based on **timeseries-connector.xml**, pointed to from an additional aggregation configuration file.

```
<config>
  <!-- The name or IP of the Backend host. -->
  <host>localhost</host>
  <!-- The port of the Backend server that the connector should use. -->
  <port>2000</port>
  <!-- The buffer size for the connection. -->
  <bufferSize>32768</bufferSize>
</config>
```

- **host:** The name or IP of either the *APG* Backend host, or the host where the Socket collector resides (used when outputting time series raw values that you want to manipulate through the standard *APG* raw values flow).
- **port:** If, for example, the destination is to be the backend directly, then the port set in this file must be the same as the Backend is listening on, as configured in: `<APG>/Backends/APG-Backend/<instance name>/conf/socketinterface.xml`. The Backend default port is 2000.

If, for example, you want to send the raw values through some filters, the destination must be a Socket collector. The Socket collector is declared in the **collecting.xml** file located at `<APG>/Collecting/Collector-Manager/<instance name>/conf`, which in turn points to the **socketcollector.xml** file located in the same directory. The port you set in the **timeseries-connector.xml** file must match the listening port configured in the **socketcollector.xml** file.

- **bufferSize:** The buffer size used for the Cache connector.

4

Aggregation Components

This section describes the value renderer, which determines the aggregation type applied to a defined field in order to combine events. It also describes the mapping engine and mappers, which determine the property set of entering events, and the concatenated key of these properties, therefore determining which events are aggregated together.

1 Value renderer

Below is the list of currently available value renderers and their usage. The aggregation is performed on the value indicated for the aggregation-value-list tags in the Event Aggregator config file. Which value renderer to use is indicated for these same tags in the type attribute. For example, the code below indicates that the SumRenderer value renderer will aggregate the value for Bytes for the incoming events.

```
<aggregation-value-list name="ifOut" unit="byte" type="SumRenderer">BYTES</aggregation-value-list>
```

To see this code in context see [Event Aggregator Configuration](#) on page 8.

1.1 AverageRenderer

This renderer will return the average of the defined field over the aggregation period.

1.2 CountRenderer

This renderer will count the entering events with the defined field over the aggregation period.

1.3 LastRenderer

This renderer will only keep the latest value of the defined field for the events over the aggregation period.

1.4 MaxRenderer

This renderer will only keep the maximum value of the defined field for the events over the aggregation period.

1.5 MinRenderer

This renderer will only keep the minimum value of the defined field for the events over the aggregation period.

1.6 SumRenderer

This renderer will return the sum of the defined field for the events over the aggregation period.

2 Mapping engine

The following mapping engine is currently available:

2.1 DefaultMappingEngine

This mapping engine will create an aggregation key out of the concatenation of the keys generated by each of its mappers. So each aggregate created from a set of events over the aggregation period will have the same properties and values. Only those properties selected and formatted by the mappers are preserved in the output events/time series.

3 Mapper

The following is the list of currently available property mappers and their usage:

Note: For every mapper that can have a pass and group parameters, the group parameter must be define as soon as at least one pass is defined.

3.1 BitMapper

This mapper provides a way to retrieve a fragment of a field that contains multiple values in an integer format. Here's the mandatory parameters for this mapper:

1. start: Specify the inclusive start index of the bit in the data (LSB = 1).
2. end: Specify the inclusive end index of the bit in the data (LSB = 1).

Configuration example:

```
<mapper-config-list>
  <mapper>BitMapper</mapper>
  <mapping-key-list property-name="mpls">MPLS_LABEL_1</mapping-key-list>
  <param-list key="start">4</mapping-key-list>
  <param-list key="end">32</mapping-key-list>
</mapper-config-list>
```

3.2 IntMapper

This mapper provides a simple way to retrieve a list of integer properties in the event base data.

It can also regroup the values outside a list of specific values or ranges. Here's the optional parameters:

1. pass: Can be defined more than once. Can be either a specific value to accept (Ex: 25) or a range of accepted values (Ex: -10/40).
2. group: Specify the value of the property for the event outside the list of specific values or ranges.

Configuration example:

```
<mapper-config-list>
  <mapper>IntMapper</mapper>
  <mapping-key-list property-name="callstatus">terminationCause</mapping-key-list>
  <param-list key="pass">10/20</param-list>
  <param-list key="group">Failed</param-list>
</mapper-config-list>
```

3.3 IPFilterMapper

This mapper provides a powerful way to collect the details of the desired address while aggregating all the others from a field containing an IP address (both IPv4 and IPv6) in a byte array format. Here's the optional parameters for this mapper:

1. detailed: Specify a single address or a subnet that won't be filtered.
2. subnet: Specify a subnet that will pass the filter, while every single IP in it will be aggregated into the subnet key. A mask is mandatory for this option.
3. grouped: Specify the value of the property for a grouped address to isolate it in the APG report. If the output value is raw, this value must be numeric.
4. output: Specify the if the formatted event will save the IP in its raw bytes (value = raw) format or in a readable format (value = string). By default, the output will be readable. This field have no purpose for raw values output.

Note: When a detailed or subnet parameter is defined, it's mandatory to add the parameter grouped.

Configuration example:

```
<mapper-config-list>
  <mapper>IPFilterMapper</mapper>
  <mapping-key-list property-name="ip">IPV4_SRC_ADDR</mapping-key-list>
  <param-list key="grouped">PublicNet</param-list>
  <param-list key="detailed">192.168.1.0/8</param-list>
  <param-list key="detailed">10.69.30.25</param-list>
  <param-list key="subnet">192.168.0.0/8</param-list>
</mapper-config-list>
```

3.4 MaskMapper

This mapper provides the aggregation key for an IPv4 address and a mask in slash or dot notation. This mapper requires both of these mapping keys:

1. ip: Specify which field contains the IP address in an integer format and the resulting property will be a dotted IP address. The property name of this field will be the one used for the subnet property (Ex: if the property name is `foo`, the property will be **foo:10.128.2.0/24** if the mask is 255.255.255.0).
2. mask: Specify what hardcoded mask to use or which field contains the mask. The format of both these cases can be either a slash or dot notation. The format for a hardcoded mask in slash notation must start with the slash (/) but if this notation is contained in a field, only the number must be present.

Configuration example:

```
<mapper-config-list>
  <mapper>MaskMapper</mapper>
  <mapping-key-list key="ip" property-name="ip">IPV4_SRC_ADDR</mapping-key-list>
  <mapping-key-list key="mask" property-name="mask">SRC_MASK</mapping-key-list>
</mapper-config-list>

<mapper-config-list>
  <mapper>MaskMapper</mapper>
  <mapping-key-list key="ip" property-name="ip">IPV4_DST_ADDR</mapping-key-list>
  <mapping-key-list key="mask" property-name="mask">/16</mapping-key-list>
</mapper-config-list>

<mapper-config-list>
  <mapper>MaskMapper</mapper>
  <mapping-key-list key="ip" property-name="ip">IPV4_NEXT_HOP</mapping-key-list>
  <mapping-key-list key="mask" property-name="mask">255.255.255.0</mapping-key-list>
</mapper-config-list>
```

3.5 StringMapper

This mapper provides a way to retrieve a list of string properties in the event base data.

It can also regroup the values that aren't matched by regexes. Here's the optional parameters:

1. pass: Can be defined more than once. Must be a valid Java regex (Ex: [0-9\w]+exe.*).
2. group: Specify the value of the property for the events that don't match any of the pass regexes.

Configuration example:

To save memory, it is possible to use an *UTF8StringMapper* instead of a *StringMapper*. It works exactly the same, but saves strings using UTF-8 instead of Java's default UTF-16. For specific cases (e.g. Netflow), where strings are already UTF-8 encoded internally, it is possible to set the *use-bytes* parameter to *true*, which avoids a charset conversion.

```
<mapper-config-list>
  <mapper>StringMapper</mapper>
  <mapping-key-list property-name="number">dialledDigits</mapping-key-list>
  <param-list key="pass">(514|450|438)\d*</param-list>
  <param-list key="group">Elsewhere</param-list>
</mapper-config-list>
```

3.6 RegexMapper

This mapper provides a way to retrieve parts of a string property via a group regex in the event based data. Each mapping key will define a property and the field to match against the regex followed by *:\$* and the group number inside the given regex. Here's the mandatory parameters for this mapper:

1. regex: The regex that the field must match. It must be a valid Java regex with groups (Ex: \+?1?(\d3)\d*)

It can also regroup the parts that aren't matched by regexes. Here's the optional parameters:

1. pass: Can be defined more than once. Must be a valid Java regex (Ex: 514|450|438).
2. group: Specify the value of the property for the parts that aren't matching any of the pass regexes.

Note: The pass parameters will be applied on each group extracted from the regex so unless the pass parameters can be applied to each of them, multiple regex mapper must be defined for the same field.

Configuration example:

```
<mapper-config-list>
  <mapper>RegexMapper</mapper>
  <mapping-key-list property-name="region">callingNumber:$1</mapping-key-list>
```

```
<param-list key="regex">\+\d(\d{3})\d{7}</param-list>
<param-list key="pass">514|450|438</param-list>
<param-list key="group">Suburb</param-list>
</mapper-config-list>
```

5

Configuration Examples

This section includes full configuration examples for a Netflow time series output, and a Broadsoft VoIP events output.

1 Netflow Time Series Output

```
<aggregation-thread>2</aggregation-thread>
<aggregation-period>240</aggregation-period>
<timestamp-field>FLOW_TMST</timestamp-field>
<destination-config output-type="raw-values">
  <params-list key="group">group</params-list>
  <params-list key="source">Netflow</params-list>
  <params-list key="config-file">conf/timeseries-connector.xml</params-list>
</destination-config>
```

The above configuration fragment specifies that every aggregated data will span over a period of four minutes by two aggregators and the timestamp field on each event will be contained in a field named FLOW_TMST. The Event Aggregator will generate *APG* raw values with the collecting group *group*

and the source Netflow. Finally, the configuration file for the backend location is `./conf/timeseries-connector.xml`.

```
<aggregation-parameter-list aggregator-name="Source">
  <aggregation-value-list name="ifOutOctets" unit="byte" type="SumRenderer">BYTES</
aggregation-value-list>
  <mapping-engine-config>
    <mapping-engine>DefaultMappingEngine</mapping-engine>
    <mapper-config-list>
      <mapper>IPFilterMapper</mapper>
      <mapping-key-list property-name="device">ROUTER_SRC</mapping-key-list>
    </mapper-config-list>
    <mapper-config-list>
      <mapper>IPFilterMapper</mapper>
      <mapping-key-list property-name="ip">IPV4_SRC_ADDR</mapping-key-list>
      <param-list key="grouped">PublicNet</param-list>
      <param-list key="detailed">10.0.0.0/8</param-list>
      <param-list key="detailed">192.168.0.0/16</param-list>
    </mapper-config-list>
    <mapper-config-list>
      <mapper>IntMapper</mapper>
      <mapping-key-list property-name="port">SRC_PORT</mapping-key-list>
      <mapping-key-list property-name="proto">PROTOCOL</mapping-key-list>
    </mapper-config-list>
  </mapping-engine-config>
</aggregation-parameter-list>
```

The above configuration fragment specifies that it will contain an aggregator named `Source` that will sum the values of `BYTES` to create the value `ifOutOctets`. These values will be aggregated by four different fields with the `DefaultMappingEngine`. The first field defining the `device` will be taken from `ROUTER_SRC` and will be parsed by the `IPFilterMapper` that will convert the field into a string IP address. The property `ip` will also be extracted via an `IPFilterMapper` from the field `IPV4_SRC_ADDR` with only the addresses from the `10.0.0.0/8` and `198.168.0.0/16` subnets detailed. Finally, the properties `port` and `proto` will be extracted with an `IntMapper` from the fields `SRC_PORT` and `PROTOCOL`.

2 Broadsoft Event Output

```
<aggregation-thread>1</aggregation-thread>
<aggregation-period>240</aggregation-period>
<timestamp-field format="ms">releaseTime</timestamp-field>
<destination-config output-type="events">
  <params-list key="source">Broadsoft</params-list>
</destination-config>
```

The above configuration fragment specifies that every aggregated data will span over a period of four minutes by a single aggregator and the timestamp field on each event will be contained in a field named `releaseTime` and formatted in milliseconds. The Event Aggregator will generate events with the static property source `Broadsoft`.

```
<aggregation-parameter-list aggregator-name="Call_Status">
  <aggregation-value-list name="count" type="CountRenderer" unit="">terminationCause</
aggregation-value-list>
  <mapping-engine-config>
    <mapping-engine>DefaultMappingEngine</mapping-engine>
    <mapper-config-list>
      <mapper>IntMapper</mapper>
      <mapping-key-list property-name="callstatus">terminationCause</mapping-key-list>
      <param-list key="pass">16</param-list>
      <param-list key="group">Failed</param-list>
    </mapper-config-list>
  </mapping-engine-config>
</aggregation-parameter-list>

<aggregation-parameter-list aggregator-name="Region_Calling">
  <aggregation-value-list name="outCount" type="CountRenderer" unit="">terminationCause</
aggregation-value-list>
  <mapping-engine-config>
    <mapping-engine>DefaultMappingEngine</mapping-engine>
    <mapper-config-list>
      <mapper>RegexMapper</mapper>
      <mapping-key-list property-name="region">callingNumber:$1</mapping-key-list>
      <param-list key="regex">\+\d(\d{3})\d{7}</param-list>
    </mapper-config-list>
  </mapping-engine-config>
</aggregation-parameter-list>
```

The above configuration fragment specifies that it will contain an aggregator named `Call Status` that will count the entering events with the field `terminationCause` to create the value `count`. These values will be aggregated by one field with the `DefaultMappingEngine`. The field defining the `callstatus` will be taken from `terminationCause` and will be parsed by the `IntMapper` that will convert the field into an integer that will let pass the value 16 and will aggregate all the rest in the value `Failed`.

Furthermore, the last aggregation named `Region Calling` will count the entering events with the field `terminationCause` to create the value `count`. These values will be aggregated by one field with the `DefaultMappingEngine`. The field defining the `region` will be taken from the first group in the regex from the field `callingNumber`.