



EMC® Watch4net®

Version 6.3u2

Event Property Tagger Admin Guide

P/N: pending

Rev : 42709

EMC²

Copyright ©2010 - 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	iv
Where to get help	v
1 Overview	1
1 Configuration Overview	1
2 Features	2
2.1 Matching on property "keys"	2
2.2 New properties based on same "key"	2
2.3 Optional usage of default values	2
2.4 String matching, SQL patterns and regular expressions	2
2.5 Optional delete of chosen key properties after tagging	2
2.6 Refresh configuration while running	3
2 Configuration	5
1 Event Processing Manager configuration	5
2 <i>Event Property Tagger</i> configuration	6
2.1 Configuration file	6
2.2 Input file	9
3 Accessors	10
3.1 JDBC Accessor	12
3.2 File Accessor	13
3.3 Static Accessor	14
3.4 Web Service Accessor	15
3.5 Property Accessor	18

3 Usage	21
1 Examples	21
1.1 Using default	21
1.2 Using regular expressions	23
1.3 Using SQL patterns	25
1.4 Using ranges	27
1.5 Deleting properties	28
1.6 Renaming properties	30
1.7 Using more than one input file	31
2 Mutable and Immutable Events	32
3 General Concepts on Matching Order	34

Preface

The purpose of this guide is to provide information about the *Event Property Tagger* and the way to use it. This guide includes all necessary information to configure and run the *Event Property Tagger*.

Intended Audience

This guide is intended to be read by any administrator who will need to add new properties to events based on the properties these events have.

Prerequisites

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

Document Organization

1. Overview	This section gives an overview of the <i>Event Property Tagger</i> .
2. Configuration	This section explains how to configure this processing element.
3. Usage	This section describes how to use this processing element.

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.

1

Overview

The *Event Property Tagger* has been designed to add new properties to events based on their existing fields.

1 Configuration Overview

Follow the workflow steps below to configure the *Event Property Tagger* and its complement elements:

1. Refer to the [Features](#) below to gain an understanding of the module.
2. Configure other processing elements preceding the *Event Property Tagger* in the flow. See corresponding module documentation.
3. Configure the **processing.xml** file to declare an *Event Property Tagger* for the Event Processing Manager (declare additional instances if necessary). See [Event Processing Manager configuration](#) on page 5.
4. Configure the *Event Property Tagger* configuration file (configure additional instances of the file if using multiple *Event Property Taggers*). See [Configuration file](#) on page 6.
5. Configure one or more input files, which help to determine, in conjunction with the *Event Property Tagger*'s configuration file, what properties to add, change, or delete. See [Input file](#) on page 9.

6. Restart the Event Processing Manager.

You do not have to restart the Event Processing Manager again if you make changes to the configuration file, or the input files, as these will be refreshed according to the value for the refresh tag indicated in the configuration file (see [page 7](#)).

2 Features

2.1 Matching on property "keys"

Multiple properties can be used as a "key" to add new properties. For example, it is possible to add the `admin` property with the value `Bob` for all events that have the property `devtype` set to `host` AND `parttype` set to `interface`.

2.2 New properties based on same "key"

A "key" can be used to add more than one new property.

2.3 Optional usage of default values

It is possible to define default values to use if there is no "key" that matches the properties of the event.

2.4 String matching, SQL patterns and regular expressions

It is possible to use SQL patterns or regular expressions instead of string comparison to define a "key". For example, you can use the SQL pattern `w4n-%` for the property `device` to matches all device values that begin with `w4n-`.

2.5 Optional delete of chosen key properties after tagging

It is possible to delete chosen properties used in the "key" after the tagging is done only in the case that the incoming event was mutable.

Note: Whether or not an event is mutable is determined by its source when it is first captured and processed.

Events that are immutable cannot have their properties deleted or modified, nor can they have properties added to them. *APG* can compensate for this behavior as explained in [Mutable and Immutable Events](#) on page 32.

For example, you may want to delete from the event a property that was used only to create a new property and that is not needed for reporting.

Note: If the deleted property already exists in the backend, it will not be deleted from the backend.

2.6 Refresh configuration while running

It is possible to reload the configuration file and its input file when they change. The module will verify for changes every configured amount of time.

2

Configuration

1 Event Processing Manager configuration

This module must be declared in the Event-Processing-Manager <APG>/Event-Processing/Event-Processing-Manager/<instance name>/conf/processing.xml file in order to be used.

```
<!-- Events Property Tagger -->  
<processing-element name="EPT" config="Event-Property-Tagger/Default/conf/event-property-tagger.xml" data="Next" />
```

Each *Event Property Tagger* instance must be declared with its own processing-element tags with any name as long as it is unique.

2 Event Property Tagger configuration

2.1 Configuration file

The **event-property-tagger.xml** configuration file is needed to run the module. It is located at: <APG>/Event-Processing/Event-Property-Tagger/<instance name>/conf.

```
<?xml version="1.0" encoding="UTF-8"?>
<property-tagging-filter-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter
    property-tagging-filter-config.xsd"
  xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">
  <refresh unit="hours">1</refresh>
  <files>
    <text-file path="input.data" encoding="UTF-8">
      <!-- The field separator for the input file -->
      <field-separator>,</field-separator>
      <!-- The quoting character. Double it to escape. -->
      <field-quoting>"</field-quoting>

      <!-- The default symbol for the input file. To use with key properties only. -->
      <default-symbol>**</default-symbol>
      <!-- The null symbol for the input file. Means property not there. To use with key
properties only. -->
      <null-symbol>@@</null-symbol>

      <!-- Use to insert property value for new properties. To use with new properties
only.-->
      <property-insertion start="[" end="]" />

      <key-properties>
        <key-property delete-after-use="false" string-type="string" decoder="IP">
src_addr</key-property>
        <key-property delete-after-use="false" string-type="regex" decoder="INT">
src_port</key-property>
        <!-- more key-property can be added -->
      </key-properties>
      <new-properties>
        <new-property encoder="IP">failover_server</new-property>
        <new-property encoder="INT">system_id</new-property>
        <!-- more new-property can be added -->
      </new-properties>
    </text-file>
  </files>
</property-tagging-filter-config>
```

```
        </new-properties>
    </text-file>
    <!-- more text-file can be added -->
</files>
</property-tagging-filter-config>
```

The `refresh` tag is mandatory. The unit can be days, hours, minutes or seconds. The value must be a positive integer. This amount of time indicates to the module when to verify for file modifications and reload them when needed. Setting the value to 0 will shut down the automatic refresh. A restart of the module with a new configuration file without 0 for the refresh value will be needed to reenale it. It is not recommended to choose a low refresh rate (under 5 minutes), because loading configurations slows down the processing.

The `file` tag is mandatory. It contains the configuration for each input file.

There is one `text-file` for each input file. The path of the input file is mandatory. The module will scan input files in the order they appear in the configuration file. The `encoding` attribute is optional. Its default value is UTF-8.

The `field-separator` tag is mandatory. It sets the separator used in the input file to split fields from a line. It can be more than one character. Be aware that if you put a space or any other whitespace character between the opening and the closing tag, these will be used, taken as is, to split the fields.

The `field-quoting` tag is optional and defaults to ". It sets the quoting string used in the input file to quote fields on a line. Quoting field is optional but you can use it only to escape special characters and the field separator. Double the quotes to escape it if quotes appear in the values themselves.

The `default-symbol` is mandatory. It may be empty if no default symbol is needed. You should not use a symbol that can conflict with property names, SQL patterns, or regular expressions. It is used to indicate everything else that has not yet been matched. For more information on how different value types are processed when there are one or more default-symbols used, see [General Concepts on Matching Order](#) on 34.

The `null-symbol` is optional. It may be empty if no null symbol is needed. You should not use a symbol that can conflict with property names, SQL patterns or regular expressions. It is used to signify a "the property does not exist" symbol for `key-property`. It can also be used to prevent a new property

from being created when placed in a `new-property` column in the data file.

The `property-insertion` is optional. It is used only for `new-properties`. It allows the insertion in the value of a new property, the value of another property, using the target property's name. For example `server-[type]` results in `server-mail` assuming the property `type` has the value `mail`.

The `key-properties` tag is mandatory. It contains all properties that, all together, are the "key".

The `key-property` tag defines a property that will be used to compose the "key". If no `key-property` is defined the new properties will be added to each event processed by the module. There is no limit to the number of key properties defined, while the order in which they appear in the file must match the columns of the input file. The value represents the name of a property to search in the event.

The `delete-after-use` attribute is optional and false by default. If set to true, the property will be deleted from the processed event AFTER all the filtering is done. That means the property can be reused in the configuration file to create other new properties.

The `string-type` attribute defines the type of the value. It can be a `string`, an `sqlpattern`, a `regex` or a `range`. Using only the `string` type makes the module run faster because it does not have to check each line for each property which it does with `sqlpatterns`, `regexes` and `ranges`.

The `decoder` attribute is optional and defines the kind of content that the field will have and it will decode it in a readable format (ex: the IP decoder will take a raw IP address in a flow event and will translate it to the dotted format). The possible decoders are `IP` (both IPv4 and IPv6), `INT` and `STRING` which is the default decoder if omitted.

The `new-properties` tag is mandatory. It contains all new properties to add to events that will match a key.

The `new-property` tag defines a new property to add to an event that will match the "key". The indicated value becomes the name of the new property in the output. At least one tag is needed. There is no limit to the number of new properties. The order they appear in the configuration must match the order their values appear in the input file.

The `encoder` attribute is optional and defines the kind of content added in the event for the field (ex: the IP encoder will take a dotted formatted address and insert it in his numeric format). The possible encoders are `IP`, `INT`, and `STRING`: which is the default encoder if the `encoder` attribute is omitted. The IP encoder supports IPv4 in dot notation and IPv6 in hexadecimal notation. If the encoding field is an IPv6 address, it will insert two numeric fields if the conversion is successful and the fields will have the suffix `_0` and `_1`.

2.2 Input file

The input file uses a "chosen string" as a separator for the values it lists. That means that CSV files are supported if in the configuration file the `field-separator` is set to `,`. Fields with an embedded "chosen strings" (in this case `,`) must be delimited with double-quotes (`"`). Blank spaces before and after will be trimmed if they are not embedded in double-quotes. If you wish to use a double-quotes character (`"`) in one of the fields, the whole field must be surrounded by double-quote characters and each of the embedded double-quotes characters must be doubled with another double-quotes. So, if you wish to have the following field:

```
This is a "quoted" field
```

You must quote the whole field like the following:

```
"This is a ""quoted"" field "
```

This represents an input file that can be used with the previous configuration file.

```
192.168.1.25,69*,192.168.5.69,25  
192.168.1.69,25*,192.168.5.25,69
```

In this case, the first column represents the `src_addr` values, the second column the `src_port` values, the third column the `failover_server` values and the fourth column the `system_id` values. Referring to the configuration file (see page 6), the two first columns represent the "key" and the two last the new properties to add to the event.

The column order depends on the configuration file. The key properties are always in the first columns and the new properties are always in the last columns.

```
KeyProperty1,KeyProperty2,...,KeyPropertyN,NewProperty1,NewProperty2,...,NewPropertyM
```

For example, we have an event that has a property named `src_addr` set to `192.168.1.25`. If the module also matches on the second entry, then the module adds the new properties `failover_server` and `system_id`.

If a line does not match with key properties and new properties, it will be ignored and a log entry will be created to warn you about it.

Note: An event could have other event properties besides those used for the key and for the matching.

These properties are simply ignored in the *Event Property Tagger*, but are preserved in the events themselves.

3 Accessors

It is possible to add some accessors into the configuration file. Accessors are able to fetch additional data from various sources. If one or more accessors are defined, previous data written in the `text-file` will be overwritten by the defined accessors. If no accessors are defined, data written in the `text-file` will be used.

The accessors are defined after the `text-file` tag. Here is an example:

```
<property-tagging-filter-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter_Property-
tagging-filter-config.xsd"
  xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">

  <!-- This is the configuration of the event property tagger -->
  <!-- Please note that the data property flat is mandatory for this -->
  <refresh unit="minutes">60</refresh>
  <files>

    <text-file path="conf/exporters.csv">
      <accessor accessor-class="JDBCAccessor">
        <parameter name="url">jdbc:mysql://localhost:53306/apg</parameter>
        <parameter name="driver">com.mysql.jdbc.Driver</parameter>
        <parameter name="user">apg</parameter>
        <parameter name="password">watch4net</parameter>
        <parameter name="query">
          SELECT DISTINCT ip, device FROM data_property_flat WHERE ip IS NOT NULL and
devtype='Router';
        </parameter>
      </accessor>
      <accessor accessor-class="StaticAccessor">
        <parameter name="line">**, [ROUTER_SRC]</parameter>
      </accessor>

      <field-separator>,</field-separator>
    </text-file>
  </files>
</property-tagging-filter-config>
```

```
<field-quoting>"</field-quoting>
<default-symbol>**</default-symbol>
<property-insertion start "[" end=""]"/>
<key-properties>
  <key-property delete-after-use="false" string-type="string" decoder="IP">
ROUTER_SRC</key-property>
</key-properties>
<new-properties>
  <new-property>ROUTER_SRC_NAME</new-property>
</new-properties>
</text-file>

</files>

</property-tagging-filter-config>
```

In this case, the text file `exporters.csv` will be overwritten by the data coming from the JDBC accessor. The data of the text-file will be refreshed each X times, depending on the `refresh` parameter. (See refresh parameter on page 7)

In case a failure happen in one of the defined accessors then the data of the `text-file` will be used. For example, if the JDBC accessor failed to fetch some data from the target database because it was down, the previous data will be used. This will happen even if the configuration have multiple accessors defined and only one of them failed.

If multiple accessors are defined, they will be executed in sequential order. This is important because the default values should always be defined in the last accessor.

Note: Make a backup of your configuration before using the accessors for the first time. The data contained in your previous csv file could be overwritten if the accessors are not configured properly.

Each accessor uses the same configuration tags. The `accessor-class` tag defines the java class of the accessor, it can also be the fully-qualified class name.

3.1 JDBC Accessor

The JDBC accessor is used to fetch data from a database using a JDBC driver. By default the drivers for a MySQL, Oracle and SQL Server database are provided. Any other JDBC driver is supported, but the driver's library must be manually copied in the lib folder of the EPT. (ie. <APG>/Event-Processing/Event-Property-Tagger/Default/lib)

Configuration example:

```
<accessor accessor-class="JDBCAccessor">
  <parameter name="url">jdbc:mysql://localhost:53306/apg</parameter>
  <parameter name="driver">com.mysql.jdbc.Driver</parameter>
  <parameter name="user">apg</parameter>
  <parameter name="password">watch4net</parameter>
  <parameter name="query">
    SELECT a , b, FROM tableA
  </parameter>
</accessor>
```

The url parameter is mandatory. The database URL.

The driver parameter is mandatory. The java class name of the JDBC driver.

The user parameter is mandatory. The database user.

The password parameter is mandatory. The user's password.

The query parameter is mandatory. The SQL query used to fetch data from the database. The number of column in the query must match the number of key-property + the number of new-property. The order of the column in the query must matches the order of the key-property and new-property.

For example, for the following configuration:

```
<key-properties>
  <key-property delete-after-use="false" string-type="string">device</key-property>
  <key-property delete-after-use="false" string-type="string">parttype</key-property>
  <key-property delete-after-use="false" string-type="string">part</key-property>
</key-properties>
<new-properties>
```

```
<new-property>desc1</new-property>
<new-property>desc2</new-property>
</new-properties>
```

The SQL query should contains 5 columns, where the first should correspond to the device key, the second to the parttype key, etc. and the last to the desc2 value. Note that the column name is not important.

The resulting query should look something like this:

device1	interface	eth0	desc1	desc2
device1	interface	eth1	description1	description 2

The JDBC accessor uses the `field-quoting` tag. For example, if the quoting tag is set to `"` then the resulting data will look like this in the `text-file` (*Note that the field-seperator is equal to `;` in this case*):

```
"device1";"interface";"eth0";"desc1";"desc2"
"device1";"interface";"eth1";"description1";"description_2"
```

If a quote symbol is present in one of the value, it will be doubled so it can be escaped.

3.2 File Accessor

The file accessor is used to fetch data from a text file. The format of the data is determine by the `field-separator` and `field-quoting` parameter. For example, if the `field-quoting` is equal to `"` and the `field-separator` is equal to `,`, then the data should look like this:

```
"device1","interface","eth0","desc1","desc2"
"device1","interface","eth1","description1","description_2"
```

Configuration example:

```
<accessor accessor-class="FileAccessor">
  <parameter name="field-separator">,</parameter>
  <parameter name="field-quoting">"</parameter>
  <parameter name="encoding">UTF-8</parameter>
  <parameter name="path">conf/default-values.csv</parameter>
</accessor>
```

The `field-separator` parameter is optional. It is the separator characters of the data file. If it's not set, then it will be equal to the `field-separator` parameter of the `text-file`.

The `field-quoting` parameter is optional. It is the quoting character of the data file. If it's not set, then it will be equal to the `field-quoting` parameter of the `text-file`.

The `encoding` parameter is optional. It is the encoding of the data file. If it's not set, then it will be equal to the `encoding` parameter of the `text-file`.

The `path` parameter is mandatory. It is the path to the data file, it can be a relative or absolute path.

3.3 Static Accessor

The Static accessor is used when you only have a few line of data. Instead of having a file accessor the data can be directly put in the configuration. It is useful for default values. Like the file accessor, the format of the data is determine by the `field-separator` and `field-quoting` parameter.

Configuration example:

```
<accessor accessor-class="StaticAccessor">
  <parameter name="field-separator">,</parameter>
  <parameter name="field-quoting">"</parameter>
  <parameter name="line">**,**,eth0,desc1,desc2</parameter>
  <parameter name="line">**,**,**,description1,description 2</parameter>
</accessor>
```

In the above example the `default-symbol` is equal to `**`.

The `field-separator` parameter is optional. It is the separator characters of each line. If it's not set, then it will be equal to the `field-separator` parameter of the `text-file`.

The `field-quoting` parameter is optional. It is the quoting character of each line. If it's not set, then it will be equal to the `field-quoting` parameter of the `text-file`.

The `line` parameter is optional. It can be repeated multiple times for each line of data.

3.4 Web Service Accessor

The Web Service Accessor is used when you want to push data to the *Event Property Tagger* from an external source. Data is sent via SOAP requests.

The Web Service Accessor uses the `webservice-gateway` for its communication, so that module must be installed. The endpoint address depends on the `webservice-gateway` configuration and the `event-processing-manager` configuration. It has this format:

`https://<host>:<port>/Event-Processing/Event-Processing-Manager/<Instance>/<EPT name>`

- `<Instance>`: This is the instance of the event processing manager.
- `<EPT name>`: This is the name of the event property tagger located in the `processing.xml` file of the event processing manager.

Endpoint example:

`https://localhost:48443/Event-Processing/Event-Processing-Manager/Default/EventPropertyTagger`

WSDL:

`https://localhost:48443/Event-Processing/Event-Processing-Manager/Default/EventPropertyTagger/?wsdl`

Configuration example:

```
<accessor accessor-class="WSAccessor">
  <parameter name="accessor-id">id1</parameter>
  <parameter name="field-separator">,</parameter>
  <parameter name="field-quoting">"</parameter>
  <parameter name="encoding">UTF-8</parameter>
  <parameter name="path">conf/ws.txt</parameter>
</accessor>
```

The `accessor-id` parameter is mandatory. It is the unique identifier of the `WSAccessor`, it must be unique within all Web Service Accessor in the configuration file. When sending SOAP request to the accessor, this `id` will be needed.

The `field-separator` parameter is optional. It is the separator characters of the data file. If it's not set, then it will be equal to the `field-separator` parameter of the `text-file`.

The `field-quoting` parameter is optional. It is the quoting character of the data file. If it's not set, then it will be equal to the `field-quoting` parameter of the `text-file`.

The `encoding` parameter is optional. It is the encoding of the data file. If it's not set, then it will be equal to the `encoding` parameter of the `text-file`.

The `path` parameter is mandatory. It is the path to the data file, it can be a relative or absolute path. This file will contain all the data that will be received via SOAP request, it acts like the file in the `FileAccessor`.

List of SOAP messages

The `add` message is used to add one or multiple lines of data to the file.

Add message example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:prop="http://www.watch4net.com/APG/Management/PropertyTaggerService">
  <soapenv:Header/>
  <soapenv:Body>
    <prop:add>
      <!-- The accessor id. -->
      <prop:accessor-id>id1</prop:accessor-id>
      <!-- The field separator characters. If not set, the field-separator of the text-file
will be used. -->
      <prop:field-separator>,</prop:field-separator>
      <!-- The field quoting character. If not set, the field-quoting of the text-file will
be used. -->
      <prop:field-quoting>"</prop:field-quoting>
      <!-- The list of lines to be added. -->
      <prop:line>"key-prop1","new-prop1"</prop:line>
      <prop:line>"key-prop2","new-prop2"</prop:line>
    </prop:add>
  </soapenv:Body>
</soapenv:Envelope>
```

The `delete` message is used to delete one or multiple lines of data to the file.

Delete message example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:prop="http://
www.watch4net.com/APG/Management/PropertyTaggerService">
  <soapenv:Header/>
  <soapenv:Body>
    <prop:delete>
      <!-- The accessor id. -->
      <prop:accessor-id>id1</prop:accessor-id>
      <!-- The field separator characters. If not set, the field-separator of the text-file
will be used. -->
      <prop:field-separator>,</prop:field-separator>
      <!-- The field quoting character. If not set, the field-quoting of the text-file will
be used. -->
      <prop:field-quoting>"</prop:field-quoting>
      <!-- The list of lines to be deleted. Only the key properties may set, if the new
properties are also present, they will not be considered. -->
      <prop:line>"key-prop1"</prop:line>
      <prop:line>"key-prop2"</prop:line>
    </prop:add>
  </soapenv:Body>
</soapenv:Envelope>
```

The replace message is used to replace the data file of WSAccessor.

Note: For the replace message, if the Content-Type is not set to application/xop+xml, then only the base64 encoded data is supported.

Replace message example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:prop="http://
www.watch4net.com/APG/Management/PropertyTaggerService">
  <soapenv:Header/>
  <soapenv:Body>
    <prop:replace>
      <!-- The accessor id. -->
      <prop:accessor-id>id1</prop:accessor-id>
      <!-- The field separator characters of the provided file. If not set, the field-
separator of the text-file will be used. -->
      <prop:field-separator>,</prop:field-separator>
      <!-- The field quoting character of the provided file. If not set, the field-quoting
of the text-file will be used. -->
```

```
<prop:field-quoting>"</prop:field-quoting>
<!-- The file which will replace the one of the WSAccessor. -->
<prop:file name="new-file.txt" size="84" encoding="utf8">cid:283430331786</prop:file>
</prop:replace>
</soapenv:Body>
</soapenv:Envelope>
```

For replacing the file, the `name`, `size` and `encoding` of the file must be supplied. The `cid` represents the Content-ID of the file in attachments.

The `clear` message is used to delete all the data of the `WSAccessor` file.

Clear message example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:prop="http://www.watch4net.com/APG/Management/PropertyTaggerService">
  <soapenv:Header/>
  <soapenv:Body>
    <prop:clear>
      <!-- The accessor id. -->
      <prop:accessor-id>id1</prop:accessor-id>
    </prop:clear>
  </soapenv:Body>
</soapenv:Envelope>
```

An external tool (`push-tagging-data`) is supplied with the *Event Property Tagger* in the `bin` folder. This tool allows you to easily send some data from an external source to the `WSAccessor` using SOAP messages. Execute the tool without arguments to see information on how to use it.

3.5 Property Accessor

The property accessor is used to fetch data (properties) from the `APG-Web-Service` module. There are a few advantages of using this accessor instead the `JDBC` accessor for getting properties. First you do not have to create the flat table. If you're using multiple databases, then `APG` web service will automatically fetch properties from all.

Note: By default the `APG Web Service` module is configured to only allows access from `localhost`. This must

be changed if the *Event Property Tagger* is installed on another server. See the APG Web Service documentation for more information.

Configuration example:

```
<accessor accessor-class="PropertyAccessor">
  <parameter name="url">http://localhost:58080/APG-WS/wsapi/db</parameter>
  <parameter name="user">admin</parameter>
  <parameter name="password">myPassword</parameter>
  <parameter name="properties">device parttype part</parameter>
  <parameter name="filter">devtype='Router'</parameter>
</accessor>
```

The `url` parameter is mandatory. It is the APG Web Service url.

The `user` parameter is mandatory. It is the user used to connected to the APG Web Service.

The `password` parameter is mandatory. It is the user's password.

The `properties` parameter is mandatory. It is the list of properties separated by a space. The number of properties must match the number of *key-property* and *new-property* defined in the *Event Property Tagger* xml configuration file.

The `filter` parameter is optional. It is the filter that will be passed to the APG Web Service to restrict the results.

Note: If the database contains null values, the row containing those values won't be fetched by the property accessor.

Usage

Once configured (page [2](#)), the *Event Property Tagger* runs with the Event-Processing-Manager.

1 Examples

1.1 Using default

The `admin` property is to be added to events. The value `alice` is needed for all events that have `host` for `devtype` and `interface` for `parttype`. The value `bob` is needed for all events that have `router` for `devtype` and `interface` for `parttype`. The value `carl` is needed for all events that have `router` for `devtype` and anything else for `parttype`. The value `ernest` is needed for all events that have anything else for `devtype` and `disk` for `parttype`. The value `david` is needed for all events that did not match for `devtype` and `parttype`.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<property-tagging-filter-config
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter
  property-tagging-filter-config.xsd"
xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">
<refresh unit="minutes">20</refresh>
<files>
  <text-file path="conf/input.data">
    <field-separator>;</field-separator>
    <default-symbol>**</default-symbol>
    <key-properties>
      <key-property delete-after-use="false" string-type="string">devtype</key-
property>
      <key-property delete-after-use="false" string-type="string">parttype</key-
property>
    </key-properties>
    <new-properties>
      <new-property>admin</new-property>
    </new-properties>
  </text-file>
</files>
</property-tagging-filter-config>
```

BAD example of an input file:

```
host; interface ; alice
router; interface ; bob
router; **; carl
**; disk; ernest
**; **; david
```

This bad example illustrates a limitation of the module. Because an event matching the line with `ernest` also matches the line with `david`, the line with `david` will always overwrite the one with `ernest`. The way to solve this problem is simply to move the line with `david` anywhere before the line with `ernest`.

Note: Lines are matched on subsequent lines only when there are multiple default-symbols used. See [General Concepts on Matching Order](#) on page 34.

Good example of an input file:

```
host; interface ; alice
router ; interface ; bob
router ; **; carl
**; **; david
**; disk ; ernest
```

Using the good example above and processing the following events:

	devtype	partype
Event 1	host	interface
Event 2	router	interface
Event 3	router	disk
Event 4	host	disk
Event 5	switch	processor

Returns these results:

	devtype	partype	admin
Event 1	host	interface	alice
Event 2	router	interface	bob
Event 3	router	disk	carl
Event 4	host	disk	ernest
Event 5	switch	processor	david

As you can see, it is possible to add ernest for the admin property because it overwrites the david entry.

1.2 Using regular expressions

It is suggested to use SQL patterns instead of regular expressions if possible because regular expressions are slower than SQL patterns.

In this example events are to be grouped using their device and devtype property. The value 7thFloor-Hosts is needed for all events whose device property begins with f17- and whose devtype is host.

The value 8thFloor-All is needed for all events whose device property begins with f18- and has anything for a devtype.

The value Routers-R is needed for all events whose device property contains within an r followed by three digits and router for devtype.

The value Others is needed for all events that have not matched for device and devtype.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<property-tagging-filter-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter
    property-tagging-filter-config.xsd"
  xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">
  <refresh unit="minutes">20</refresh>
  <files>
    <text-file path="conf/input.data">
      <field-separator>, </field-separator>
      <default-symbol>***</default-symbol>
      <key-properties>
        <key-property delete-after-use="false" string-type="regex">device</key-property>
        <key-property delete-after-use="false" string-type="regex">devtype</key-property>
      </key-properties>
      <new-properties>
        <new-property>group</new-property>
      </new-properties>
    </text-file>
  </files>
</property-tagging-filter-config>
```

Be careful with regular expressions added to the input file since there are many ways to use them, and it is easy to create one that will overwrite another. For example, if you create a line with `abc.*` and another with `.*def` and the value `abcdef` passes the filter, it will match both lines. The last match will overwrite the preceding match.

Example input file:

```
f17-.*, host, 7thFloor-Hosts
f18-.*, .*, 8thFloor-All
.*r[0-9]{3}.*, router, Routers-R
***, ***, Others
```

Using the example above and processing the following events:

	device	devtype
Event 1	f17-asd-56	host
Event 2	f17-fas-42	router
Event 3	f18-rew-rwe	host
Event 4	f1123-qwe-r342-wq	router

Returns these results:

	device	devtype	group
Event 1	f17-asd-56	host	7thFloor-Hosts
Event 2	f17-fas-42	router	Others
Event 3	f18-rew-rwe	host	8thFloor-All
Event 4	f1123-qwe-r342-wq	router	Routers-R

1.3 Using SQL patterns

SQL patterns are faster than regexes but give less control.

SQL patterns use "%" and "_" special characters only. "%" is for any characters (of any length). "_" is for any single character.

For instance: for this pattern %abc_ef, the following string will pass 123abcdef, but not this one 123abcddef.

In this example events are to be grouped using their device and devtype properties. The value 7thFloor-Hosts is needed for all events whose device property begins with f17- and devtype is host.

The value 8thFloor-All is needed for all events whose device property begins with f18- and has anything for a devtype.

The value Others is needed for all events that have not matched for device and devtype.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<property-tagging-filter-config
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter
  property-tagging-filter-config.xsd"
xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">
<refresh unit="minutes">20</refresh>
<files>
  <text-file path="conf/input.data">
    <field-separator>, </field-separator>
    <default-symbol>***</default-symbol>
    <key-properties>
      <key-property delete-after-use="false" string-type="sqlpattern">device</key-
property>
      <key-property delete-after-use="false" string-type="sqlpattern">devtype</key-
property>
    </key-properties>
    <new-properties>
      <new-property>group</new-property>
    </new-properties>
  </text-file>
</files>
</property-tagging-filter-config>

```

Be careful with SQL patterns added to the input file since there are many ways to use them, and it is easy to create one that will overwrite another. For example, if you create a line with `abc%` and another with `%def` and the value `abcdef` passes the filter, it matches both lines. The last match will overwrite the preceding match.

Example input file:

```

f17 —%, host, 7thFloor—Hosts
f18 —%, %, 8thFloor—All
***, ***, Others

```

Using the example and processing the following events:

	device	devtype
Event 1	f17-asd-56	host
Event 2	f17-fas-42	router
Event 3	f18-rew-rwe	host

Returns these results:

	device	devtype	group
Event 1	fl7-asd-56	host	7thFloor-Hosts
Event 2	fl7-fas-42	router	Others
Event 3	fl8-rew-rwe	host	8thFloor-All

Note: The escape character for SQL Patterns is the backslash '\'.

1.4 Using ranges

Ranges are used with number properties. Range configurations accepts inclusive/exclusive boundaries and decimal numbers.

For example: for this range [0;20.5[, the following string will pass 0, but not this one 20.5.

In this example a codegrp property is to be added to events with a group code property.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<property-tagging-filter-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter
    property-tagging-filter-config.xsd"
  xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">
  <refresh unit="minutes">20</refresh>
  <files>
    <text-file path="conf/input.data">
      <field-separator>,</field-separator>
      <default-symbol>***</default-symbol>
      <key-properties>
        <key-property delete-after-use="false" string-type="range">code</key-property>
      </key-properties>
      <new-properties>
        <new-property>codegrp</new-property>
      </new-properties>
    </text-file>
  </files>
</property-tagging-filter-config>
```

Example input file:

```
[0;100[,gold
```

```
[100;199], silver  
]199;300[, bronze
```

Using the example and processing the following events:

	code
Event 1	4
Event 2	199
Event 3	205

Returns these results:

	code	codegrp
Event 1	4	gold
Event 2	199	silver
Event 3	205	bronze

1.5 Deleting properties

It is possible to delete properties that were used to create new properties by setting to true the `delete-after-use` attribute in the corresponding key-property, if the incoming event is a mutable one. Otherwise, the properties are not deleted and a warning is logged. See [Mutable and Immutable Events](#) on page 32.

In this example, a property is to be deleted that is only used to create new properties. Once deleted, it will not be sent to the other processing elements.

The `departmt` property is to be created using the `group` property, then the `group` property must be deleted. Every group with value `grp-dep12-swt` has to be in `dep-12` for `departmt`. Every group with value `grp-dep12-rtr` has to be in `dep-12` for `departmt`. Every group with value `grp-dep13-hst` has to be in `dep-13` for `departmt`. Every group with value `grp-dep14-ftp` has to be in `dep-14` for `departmt`.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>  
<property-tagging-filter-config  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter  
    property-tagging-filter-config.xsd"  
  xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">  
  <refresh unit="minutes">20</refresh>  
  <files>  
    <text-file path="conf/input.data">
```

```
<field-separator>,</field-separator>
<default-symbol>***</default-symbol>
<key-properties>
  <key-property delete-after-use="true" string-type="string">group</key-property>
</key-properties>
<new-properties>
  <new-property>departmt</new-property>
</new-properties>
</text-file>
</files>
</property-tagging-filter-config>
```

Example input file:

```
"grp-dep12-swt","dep-12"
"grp-dep12-rtr","dep-12"
"grp-dep13-hst","dep-13"
"grp-dep14-ftp","dep-14"
```

Using the example and processing the following events:

	group
Event 1	grp-dep12-swt
Event 2	grp-dep12-rtr
Event 3	grp-dep13-hst
Event 4	grp-dep14-ftp
Event 5	grp-dep10-dsa

Returns these results:

	group	departmt
Event 1	-no property-	dep-12
Event 2	-no property-	dep-12
Event 3	-no property-	dep-13
Event 4	-no property-	dep-14
Event 5	grp-dep10-dsa	-no property-

As indicated, the fifth event still has its group property set after passing through the filtering because it did not match a line from the input file. As well, since there was no 'default value' configured, the fifth event has equally not had its property replaced.

1.6 Renaming properties

In this example, the values of some device properties are to be renamed.
All events with w4n-rtr-123 for device must be renamed to w4n-router-123.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<property-tagging-filter-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter
    property-tagging-filter-config.xsd"
  xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">
  <refresh unit="minutes">20</refresh>
  <files>
    <text-file path="conf/input.data">
      <field-separator>,</field-separator>
      <default-symbol>***</default-symbol>
      <key-properties>
        <key-property delete-after-use="false" string-type="string">device</key-property>
      </key-properties>
      <new-properties>
        <new-property>device</new-property>
      </new-properties>
    </text-file>
  </files>
</property-tagging-filter-config>
```

Example input file.

w4n-rtr-123,w4n-router-123

Using the example and processing the following events:

	device
Event 1	w4n-rtr-123
Event 2	w4n-swt-231

Returns these results:

	device
Event 1	w4n-router-123
Event 2	w4n-swt-231

1.7 Using more than one input file

It is possible to use multiple input files with different key properties in the module. text-files tags are scanned in the order they appear in the configuration file.

New properties subgroup and group are to be created. Groups will contain subgroups.

All events with device ending with d32 will create and set subgroup to d32. All events with device ending with d33 will create and set subgroup to d33. All events with device ending with g18 will create and set subgroup to g18. All events with other device values will create and set subgroup to x.

All events with subgroup beginning with d will create and set group to d. All events with subgroup beginning with g will create and set group to g. All events with subgroup has x will create and set group to x.

Example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<property-tagging-filter-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter
    property-tagging-filter-config.xsd"
  xmlns="http://www.watch4net.com/APG/Filter/PropertyTaggingFilter">
  <refresh unit="minutes">20</refresh>
  <files>
    <text-file path="conf/input1.data">
      <field-separator>,</field-separator>
      <default-symbol>***</default-symbol>
      <key-properties>
        <key-property delete-after-use="false" string-type="sqlpattern">device</key-
property>
      </key-properties>
      <new-properties>
        <new-property>subgroup</new-property>
      </new-properties>
    </text-file>
    <text-file path="conf/input2.data">
      <field-separator>;</field-separator>
      <default-symbol>%</default-symbol>
      <key-properties>
        <key-property delete-after-use="false" string-type="sqlpattern">subgroup</key-
property>
      </key-properties>
      <new-properties>
```

```
<new-property>group</new-property>
</new-properties>
</text-file>
</files>
</property-tagging-filter-config>
```

Example input file 1.

```
%d32,d32
%d33,d33
%g18,g18
***,X
```

Example input file 2.

```
d%,d
g%,g
x,x
```

Using the example and processing the following events:

	device
Event 1	w4n-123456-d32
Event 2	w4n-987654-d33
Event 3	w4n-987654-g18
Event 4	w4n-987654-r56

Returns these results:

	device	subgroup	group
Event 1	w4n-123456-d32	d32	d
Event 2	w4n-987654-d33	d33	d
Event 3	w4n-987654-g18	g18	g
Event 4	w4n-987654-r56	x	x

2 Mutable and Immutable Events

Immutable events are events with properties that cannot be modified, added, or deleted. Mutable events by contrast can have their properties changed, added, or removed. Whether events are mutable or immutable is determined when they are first processed by their source into the APG event processing flow (for example by the Flow-Listener). Most collected events are of the immutable type.

It is possible to change, delete, or add properties to an immutable event in APG (for example, by passing the event through the Event-Property-Tagger). When this happens the immutable event is encapsulated by a mutable event, where the immutable event's original properties and values are retained by the encapsulation, and the modified properties are applied to the mutable portion.

When changes are applied to an immutable event, the following occurs:

- A resulting mutable event is created with the property values changed.
- If there were new properties added, these are part of the mutable event.
- If there were properties deleted, these become part of the encapsulated immutable event.
- For any properties whose values are changed, their original values are stored in the encapsulated immutable event.

Typically, if an element further along in the event processing flow searches for a property whose value has been modified, the changed value is returned . But if a property no longer exists in the mutable event, it still exists in the encapsulated immutable event, so will nonetheless be returned.

So for example, if there is event E that contains the fields A and B and the *Event Property Tagger* is configured to change the value of the field A, add the field C, and delete the field B. The resulting event will be:

```
Mutable Event
A=newValue C=cValue
Encapsulated within the event, immutable
A=oldValue B=bValue
```

So elements further along in the processing chain will find “newValue” when searching for A; and “bValue” when searching for B. Since deleted event properties are still available in the event, they can be mapped in the database and therefore would be available in the Frontend. If property values have been changed however, only the new changed values will be available in the Frontend if these properties are mapped.

3 General Concepts on Matching Order

As illustrated in the examples above, event properties are matched against the input files line-by-line but there is different behavior for `sqlpatterns`, `regexes` and `ranges`, than for `strings`. As well, there are some effects on which lines of an input file events are matched against when using lines with default-symbols.

Again, as described above, for `sqlpatterns`, `regexes` and `ranges`, the properties of each incoming event are first matched against each line of the input file; typically, regular lines, that is, lines without default-symbols, appear first in the file, are treated the same, and are evaluated first. So if there are three regular lines, and the incoming event matches the first and third line, the third line is used being output with the corresponding properties into the event flow. In effect, the event breaks on the last regular line match that is made. If no match is made on these regular lines, then the incoming event is matched against any line with default-symbol characters, usually located as the last line(s), of the input file.

On the other hand, for `strings`, the incoming event's properties will be matched against regular lines, but on the first line that matches the event breaks and is not re-evaluated on subsequent lines, being sent on to the next processing element immediately. In effect, the event breaks on the first regular line match that is made. If it does not match any previous lines, and there is a line with default-symbol characters, it will then be matched against this line.

Then for all cases, `strings`, `sqlpatterns`, `regexes` and `ranges`, if an event's properties have not matched any previous lines, and there are default-symbols in the same column position on subsequent lines, and all values in the input file line before the default-symbol are the same (or there are no values before the default-symbol, as in the [Using default](#) example on page 21) the event will be re-evaluated on each of these lines.

Another example of this is the following, being the last lines of an input file, with three properties to match (this uses a string example, but is conceptually applicable to all four types):

```
Host,**,H41,Montreal  
Host,**,**,Toronto
```

For the event:

```
Host,interface,H41
```

Will match the second line to become Host,interface,H41,Toronto