



EMC® Watch4net®

Version 6.4u2

Property Tagging Filter Administration Guide

P/N: pending

Rev : 50919

EMC²

Copyright ©2008 - 2014 EMC Corporation. All rights reserved. Published in the USA.

Published October, 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 Overview	1
1 Features	1
1.1 Composed keys	1
1.2 New properties based on same "key"	1
1.3 Optional usage of default values	2
1.4 String matching, SQL patterns and regular expressions	2
1.5 Optional delete of chosen key properties after tagging	2
1.6 Refresh configuration while running	2
2 Configuration	3
1 Collector Manager configuration	3
2 <i>Property Tagging Filter</i> configuration	3
2.1 Configuration file	3
2.2 Input file	7
3 Accessors	8
3.1 JDBC Accessor	9
3.2 File Accessor	11
3.3 Static Accessor	12
3.4 Web Service Accessor	12
3.5 Property Accessor	16
3.6 Smarts domain tagging accessor	17
3.7 Smarts retrieve groups accessor	18
3.8 Sparql Accessor	20
3.9 Post processing	22

3 Usage	23
1 Examples	23
1.1 Using default	23
1.2 Using regular expression	25
1.3 Using SQL pattern	27
1.4 Using ranges	28
1.5 Delete properties	30
1.6 Renaming properties	31
1.7 Using more than one input file	32
2 General Concepts on Matching Order	34
2.1 Optimized mode (legacy)	34
2.2 Linear mode	39

Preface

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

Intended Audience

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

Prerequisites

Watch4Net's *APG* Backend and Frontend must be installed. Versions higher than 4.0 are supported.

Document Organization

1. Overview	This section gives an overview of the <i>Property Tagging Filter</i> .
2. Configuration	This section explains how to configure the filter.

3. Usage	This section describes how to use 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.

1

Overview

The *Property Tagging Filter* has been designed to add new properties to raw values based on the properties they already have.

1 Features

1.1 Composed 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 raw values that have the property `devtype` set to `host` AND `parttype` set to `interface`.

1.2 New properties based on same "key"

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

1.3 Optional usage of default values

It is possible to define default values to use if there is no "key" that match the properties of the raw value.

1.4 String matching, SQL patterns and regular expressions

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

1.5 Optional delete of chosen key properties after tagging

It is possible to delete chosen properties used in the "key" after the tagging is performed. For example, you may want to delete from the raw value a property that was used only to create a new property and that is not needed for reporting.

1.6 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 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.

```
<filter enabled="true" name="Property-Tagging-Filter" next="Backend" type="PropertyTaggingFilter" conf="Property-Tagging-Filter/Default/conf/property-tagging-filter.xml" />
```

A filter tag must be created with a custom name.

2 *Property Tagging Filter* configuration

2.1 Configuration file

The configuration file is needed to run the filter.

```

<?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">

      <!-- How to process data, default is optimized. linear is also available. -->
      <data-processing type="optimized" />

      <!-- 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. -->
      <null-symbol>@@</null-symbol>
      <!-- Match-all symbol. To use with key properties only. -->
      <match-all-symbol>%%</match-all-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">devtype</key-
property>
        <key-property delete-after-use="false" string-type="regex">device</key-property>
        <!-- more key-property can be added -->
      </key-properties>
      <new-properties>
        <new-property>admin</new-property>
        <new-property>RV_GroupName</new-property>
        <!-- more new-property can be added -->
      </new-properties>
    </text-file>
  </files>
</property-tagging-filter-config>

```

```
<!-- 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 filter when to verify for file modifications and reload them when needed. It is not recommended to choose a low refresh rate (under 5 minutes), because loading configurations slows down the filtering.

Setting the value to 0 will shut down the automatic refresh. A restart of the filter with a new configuration with a refresh value other than 0 is necessary to reenale it.

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 filter 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 `data-processing type` tag is optional. Its default type is `optimized` (Same as before). It is also possible to set it to `linear` to allow data processing line by line with no exceptions. That means no default behaviour and no performance improvement from combining similar rows.

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, it will be 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 quote to escape it in the input file.

The `default-symbol` is mandatory. It may be empty if no default symbol is needed. Be aware to not use a symbol that can conflict with a property name, SQL pattern or regular expression. 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 page 34. It can only be used for tagging `key-properties`.

The `null-symbol` is optional. It may be empty if no null symbol is needed. Be aware to not use a symbol that can conflict with a property name, SQL pattern or regular expression. It is used to indicate a "the property does not exist" symbol for a `key-property`. It can also be used to prevent the creation

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

The `match-all-symbol` is optional. It is used to always match for a key-property even if the property does not exist. It is not available for use with string string-type as its behaviour would conflict with it.

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

The `key-properties` tag is mandatory. It contains all the properties that, combined, constitute the new "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 raw value processed by the module. There is no limit about the number of key properties defined: the order in which they are presented must match the columns of the input file. The value represents the name of a property to search in the raw value. The `delete-after-use` attribute is optional and false by default. If set to true, the property will be deleted from the processed raw value AFTER all the filtering is done. This 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 exclusively 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 `new-properties` tag is mandatory. It contains all new properties to add to raw values matching a key.

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

Note: Using the keyword `RV.GroupName`, it's possible to trigger on or modify the retention group name of the raw values. Using the keyword `RV.Value`, it's possible to trigger on or modify the variable's value of the raw values. Finally, using the keyword `RV.Variable`, it's possible to trigger on or modify the variable

id of the raw values.

2.2 Input file

The input file is a "chosen string" separated values. That means that CSV files are supported if in the configuration file the `field-separator` is set to `,`. Fields with embedded "chosen string" (in that 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 character 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.

```
host,w4n-.*,alice,hostGroup
router,zzz-.*,bob,routerGroup
```

In this case, the first column represents the `devtype` values, the second column the `device` values, the third column the `admin` values and the fourth column the `group name` values. Referring to the configuration files, the two first columns represent the "key" and the two last the new properties to add to the raw value.

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, there is a raw value that has a property named `devtype` set to `router`. If the filter also matches on the second entry, then the filter adds the new properties `admin` and changes the `group name`.

If a line does not match with key properties and new properties, it will be ignored and a log entry will be created as a warning.

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>
      <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 5)

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 PTF. (ie. `<APG>/Collecting/Property-Tagging-Filter/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-separator 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 *Property Tagging Filter* 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 collector-manager configuration. It has this format:

`https://<host>:<port>/Collecting/Collector-Manager/<CM Instance>/<PTF name>`

- <CM Instance>: This is the instance of the collector manager.
- <PTF name>: This is the name of the property tagging filter located in the `collecting.xml` file of the collector manager.

Endpoint example:

`https://localhost:48443/Collecting/Collector-Manager/Default/PropertyTaggingFilter`

WSDL:

`https://localhost:48443/Collecting/Collector-Manager/Default/PropertyTaggingFilter/?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 *Property Tagging Filter* 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 *Property Tagging Filter* 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 *Property Tagging Filter* 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.

The disableSSLValidation parameter is optional. Default value is false. If set to true when using https URL it will accept untrusted certificates.

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

3.6 Smarts domain tagging accessor

The smarts domain tagging accessor is used to fetch domain tagging data from Smarts. It does so by retrieving all the ICS.DomainTagConfiguration instances and extract the pattern property. After that, it follows the containment link of the tags property and extracts the Name and tagString properties. This accessor is static in the number of fields it returns. The return entry list is defined as follow : "domain", "pattern", "tag" where domain "ICS-Domain-" prefix is stripped.

Configuration example:

```
<accessor accessor-class="SmartsDomainTaggingAccessor">
  <parameter name="auth-mode">BROKER</parameter>
  <parameter name="host">localhost</parameter>
  <parameter name="port">426</parameter>
  <parameter name="broker-user">admin</parameter>
  <parameter name="broker-pass">changeme</parameter>
  <parameter name="domain-user">admin</parameter>
  <parameter name="domain-pass">changeme</parameter>
  <parameter name="domain">INCHARGE-SA</parameter>
</accessor>
```

The `auth-mode` parameter is mandatory. The possible values are : BROKER or DOMAIN. The broker mode means that the authentication must pass by the broker, domain will be a direct connection to the domain manager.

The `host` parameter is mandatory. It is the host or ip of the Smarts you want to connect to.

The `port` parameter is mandatory. It is the broker or domains port of Smarts.

The `broker-user` parameter is mandatory when BROKER auth mode. It is the broker user name.

The `broker-pass` parameter is mandatory when BROKER auth mode. It is the broker user's password.

The `domain-user` parameter is mandatory when DOMAIN auth mode and optional in BROKER auth mode. It is the domain user name.

The `domain-pass` parameter is mandatory when DOMAIN auth mode and optional in BROKER auth mode. It is the domain user's password.

The `domain` parameter is mandatory. It is the name of the domain to attach to.

3.7 Smarts retrieve groups accessor

The smarts retrieve groups accessor is used to extract all the hierarchical groups from Smarts to tag all the referees of the group. It does so by retrieving all the HierarchicalGroup instances and extract the ParentGroup, ChildGroups, ConsistsOf and Name properties. It will follow all the parents and childs groups to get all the groups related to one instance of an object. After that, it uses the the instance name of the reference link of the ConsistsOf property. This accessor is static in the number of fields it returns. The return entry list is defined as follow : "instance name of the referee", "group" where groups is the a concatenated list split by the separator property. When retrieving the group names, we compute the name of a group according to it's hierarchy. That means that it's name plus all it's parent will be concatenated together. This is

Configuration example:

```
<accessor accessor-class="SmartsRetrieveGroupsAccessor">
  <parameter name="auth-mode">BROKER</parameter>
```

```
<parameter name="host">localhost</parameter>
<parameter name="port">426</parameter>
<parameter name="broker-user">admin</parameter>
<parameter name="broker-pass">changeme</parameter>
<parameter name="domain-user">admin</parameter>
<parameter name="domain-pass">changeme</parameter>
<parameter name="domain">INCHARGE-SA</parameter>

<!-- Separator for the groups. Defaults to | -->
<parameter name="group-separator">|</parameter>

<!-- Separator for the parent names. Optional, defaults to unspecified. -->
<parameter name="name-separator"></parameter>

<!-- Separator for the groups. Defaults to Name -->
<parameter name="group-name-property">Name</parameter>

<!-- Separator for the groups. Defaults to Name -->
<parameter name="device-name-property">Name</parameter>

</accessor>
```

The `auth-mode` parameter is mandatory. The possible values are : BROKER or DOMAIN. The broker mode means that the authentication must pass by the broker, domain will be a direct connection to the domain manager.

The `host` parameter is mandatory. It is the host or ip of the Smarts you want to connect to.

The `port` parameter is mandatory. It is the broker or domains port of Smarts.

The `broker-user` parameter is mandatory when BROKER auth mode. It is the broker user name.

The `broker-pass` parameter is mandatory when BROKER auth mode. It is the broker user's password.

The `domain-user` parameter is mandatory when DOMAIN auth mode and optional in BROKER auth mode. It is the domain user name.

The `domain-pass` parameter is mandatory when DOMAIN auth mode and optional in BROKER auth mode. It is the domain user's password.

The `domain` parameter is mandatory. It is the name of the domain to attach to.

The `group-separator` parameter is optional. It is the string used as group separator. It defaults to the pipe (—) character.

The `name-separator` parameter is optional. It is the string used to concatenate the parent group names with. If not specified, the parent group names will not be concatenated.

The `group-name-property` parameter is optional. It is the property to use on the group instance to use as name.

The `device-name-property` parameter is optional. It is the property to use on the device instance to use as name.

3.8 Sparql Accessor

The Sparql accessor is used to fetch data from a Sparql Endpoint. This can be used to retrieve data from Resource Description Framework (RDF) repositories such as the Topology repository. The Sparql Endpoint will execute the configured query. The returned binding set will be written to the data file.

Configuration example:

```
<accessor accessor-class="com.watch4net.apg.v2.collector.plugins.propertytaggingfilter.accessor
.SparqlAccessor">
  <parameter name="url">https://localhost:48443/Backends/Topology-Service/Default/topology/
repository/sparql</parameter>
  <parameter name="user">admin</parameter>
  <parameter name="password">myPassword</parameter>
  <parameter name="disableSSLValidation">true</parameter>
  <parameter name="query">
    PREFIX srm: &lt;http://ontologies.emc.com/2013/08/srm#&gt;
    SELECT ?vplexid ?vvol ?lunwnn ?parttype ?plname
    WHERE {
      ....
    }
  </parameter>
</accessor>
```

The `url` parameter is mandatory. This is the URL of the Sparql endpoint.

The `user` parameter is optional. This is the username used to access the Sparql endpoint. Although the username is optional, it must be provided if the Sparql endpoint is configured to require authentication.

The `password` parameter is optional. This is the password used to access the Sparql endpoint. Although the password is optional, it must be provided if the Sparql endpoint is configured to require authentication.

The `disableSSLValidation` parameter is optional. If set to true, SSL validation will be disabled.

The `query` parameter is mandatory. This is the Sparql query that will be sent to the Sparql endpoint.

For example, for the following configuration:

```
<key-properties>
  <key-property delete-after-use="false" string-type="string">vplexid</key-property>
  <key-property delete-after-use="false" string-type="string">vvol</key-property>
  <key-property delete-after-use="false" string-type="string">lunwwn</key-property>
  <key-property delete-after-use="false" string-type="string">parttype</key-property>
</key-properties>
<new-properties>
  <new-property>poolname</new-property>
</new-properties>
```

the Sparql query should contains 5 columns, where the first should correspond to the `vplexid` key, the second to the `vvol` key, etc. and the last to the `poolname` value. Note that the column name is not important.

The results should look something like this:

vplexid1	vvol1	lunwwn1	parttype1	poolname1
vplexid2	vvol2	lunwwn2	parttype2	poolname2

The Sparql 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*):

```
"vplexid1";"vvol1";"lunwwn1";"parttype1";"poolname1"
```

```
"vplexid2";"vvol2";"lunwwn2";"partype2";"poolname2"
```

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

3.9 Post processing

Post processing is used to modify data obtained through accessors. It allow to apply regular expression on values (column) and for whole lines. The column number starts at 1.

```
<accessor accessor-class="StaticAccessor">
  <parameter name="line">deviceA,groupAA</parameter>
  <post-process>
    <column replacement="$1" column="2" regex="(a+)bb" />
    <!-- more columns regex -->
    <line replacement="$1,$2,$1" regex="^(.+),(.+),(.+)$" />
    <!-- more line regex -->
  </post-process>
</accessor>
```

Usage

Once configured (page [2](#)), the *Property Tagging Filter* runs with the Collector-Manager.

1 Examples

1.1 Using default

The `admin` property is to be added to raw values. The value `alice` is needed for all raw values that have `host` for `devtype` and `interface` for `parttype`. The value `bob` is needed for all raw values that have `router` for `devtype` and `interface` for `parttype`. The value `carl` is needed for all raw values that have `router` for `devtype` and anything else for `parttype`. The value `ernest` is needed for all raw values that have anything else for `devtype` and `disk` for `parttype`. The value `david` is needed for all raw values 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>

```

Example on an input file:

```

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

```

This example illustrates a limitation of the filter. A raw value having router as devtype and disk as parttype would be matched by the three last lines of the input file. Among the three, the raw value is actually processed by the line being more specialized (in this case the line with carl). See [General Concepts on Matching Order](#) on page 34 for more details on matching order.

Using the example and processing raw values like these:

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

Returns these results:

	devtype	partype	admin
Raw value 1	host	interface	alice
Raw value 2	router	interface	bob
Raw value 3	router	disk	carl
Raw value 4	host	disk	ernest
Raw value 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 expression

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

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

The value 8thFloor-All is needed for all raw values whose device property begins with f18- and for any devtype.

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

The value Others is needed for all raw values 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, 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 with the two 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 good example and processing raw values like these:

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

Returns these results:

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

1.3 Using SQL pattern

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 raw values are to be grouped using their device and devtype property. The value 7thFloor-Hosts is needed for all raw values that their device property begin with f17- and devtype is host.

The value 8thFloor-All is needed for all raw values that their device property begin with f18- and for any devtype.

The value Others is needed for all raw values 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 will match with the two 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 raw values like these:

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

Returns these results:

	device	devtype	group
Raw value 1	f17-asd-56	host	7thFloor-Hosts
Raw value 2	f17-fas-42	router	Others
Raw value 3	f18-rew-rwe	host	8thFloor-All

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

Note: `sqlpattern-ci` is the same as `sqlpattern`: these are case insensitive. Use `sqlpattern-cs` to use case sensitive matches.

1.4 Using ranges

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

For example: for this range `[0;20.5[`, the following string will pass 5, but not 21.

In this example a `codegrp` property is to be added to raw values 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 raw values like these:

	code
Raw value 1	4
Raw value 2	199
Raw value 3	205

Returns these results:

	code	codegrp
Raw value 1	4	gold
Raw value 2	199	silver
Raw value 3	205	bronze

1.5 Delete 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 `key-property` tag you want.

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.

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

The `departmt` property is to be created using the `group` property, then the `group` property must be deleted. All group with value `grp-dep12-swt` have to be in `dep-12` for `departmt`. All group with value `grp-dep12-rtr` have to be in `dep-12` for `departmt`. All group with value `grp-dep13-hst` have to be in `dep-13` for `departmt`. All group with value `grp-dep14-ftp` have 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 raw values like these:

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

Returns these results:

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

As you can see, the fifth raw value still has it's group property set after going through filtering. This is because there was no 'default value' configured, hence the fifth raw value has not had its property replaced.

1.6 Renaming properties

In this example, the values of some device properties are to be renamed.

All raw values with w4n-rtr-123 for device must be renamed has 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 raw values like these:

	device
Raw value 1	w4n-rtr-123
Raw value 2	w4n-swt-231

Returns these results:

	device
Raw value 1	w4n-router-123
Raw value 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 filter. `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 raw values with `device` ending with `d32` will create and set `subgroup` to `d32`. All raw values with `device` ending with `d33` will create and set `subgroup` to `d33`. All raw values with `device` ending with `g18` will create and set `subgroup` to `g18`. All raw values with other `device` values will create and set `subgroup` to `x`.

All raw values with `subgroup` beginning with `d` will create and set `group` to `d`. All raw values with `subgroup` beginning with `g` will create and set `group` to `g`. All raw values 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 raw values like these:

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

Returns these results:

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

2 General Concepts on Matching Order

2.1 Optimized mode (legacy)

As illustrated in the examples above, raw value’s properties are matched against the input files line-by-line. Lines are however not tested in the order they appear but are sorted using an implicit priority. In most of the case, this modified order does not affect the results, but it is good to keep it in mind.

Here is a technical example of how it works.
Example configuration file:

```
[...]
<text-file path="conf/input1.data">
  <field-separator>,</field-separator>
  <default-symbol>**</default-symbol>
  <match-all-symbol>%</match-all-symbol>
```

```
<key-properties>
  <key-property delete-after-use="false" string-type="sqlpattern">k1</key-property>
  <key-property delete-after-use="false" string-type="sqlpattern">k2</key-property>
  <key-property delete-after-use="false" string-type="sqlpattern">k3</key-property>
</key-properties>
<new-properties>
  <new-property>n1</new-property>
</new-properties>
</text-file>
[...]
```

With this data file.

```
%%,%%,%%, A
11,%%,33, B
22,11,33, C
%%,11,%%, D
```

When loading the configuration, the data file is read line by line.

%%,%%,%%,A

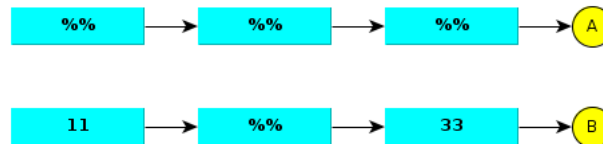
Will be represented internally in a tree like that:



Where each node level is a column from the data file.

Then the second line:

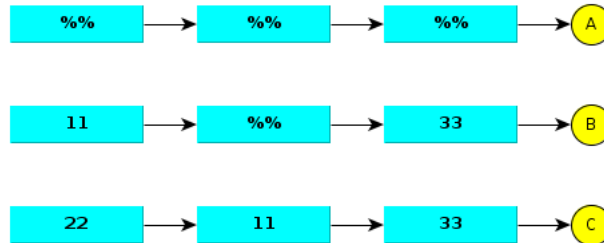
11,%%,33,B



A new path will be created.

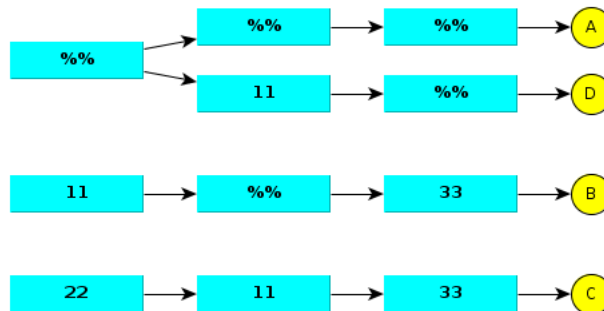
Same for third line:

22,11,33,C



And then the fourth line:

%%,11,%%,D



Here is the catch; because the fourth line first column shares the same value as the first line first column, its path is shared with the first line. That way the common expression will only be checked once.

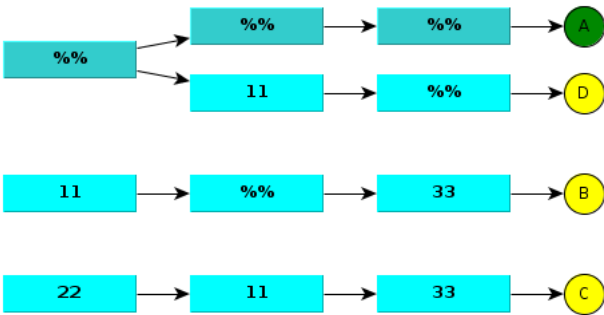
The optimized mode should be used in most cases as it offer more performance. The only cases when we would not use it is when received data may match several lines (using regex or sqlpattern

expressions).

Now we receive a value with these properties:

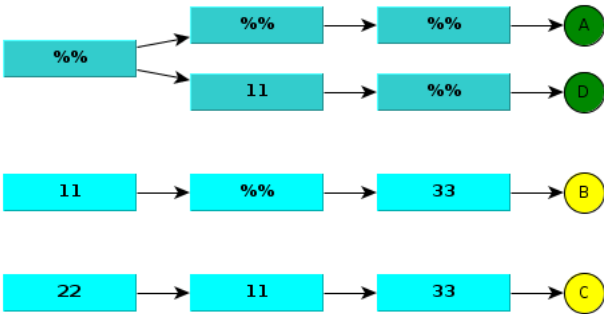
	k1	k2	k3
Raw value 1	22	11	33

Tree paths are followed step by step:



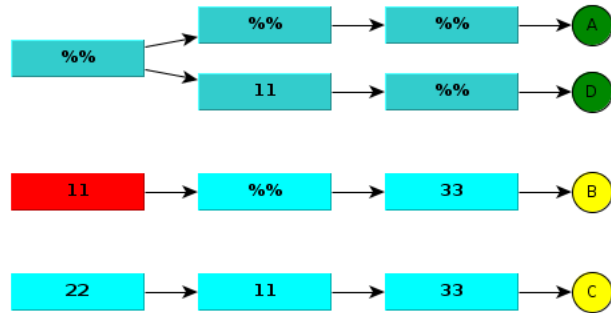
The first path matched so the new property `n1` is created and assigned value A.

Then the second path:



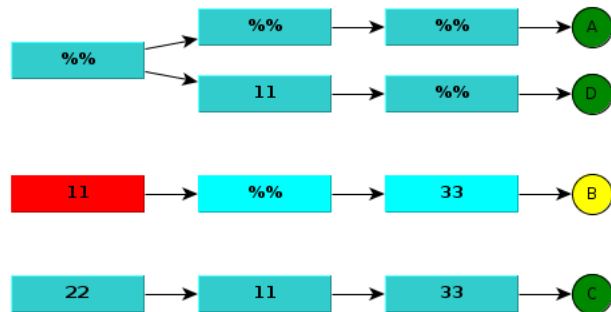
It also matched. So the value of `n1` is now D.

Then the third path:



Not matched.

Finally the fourth path:



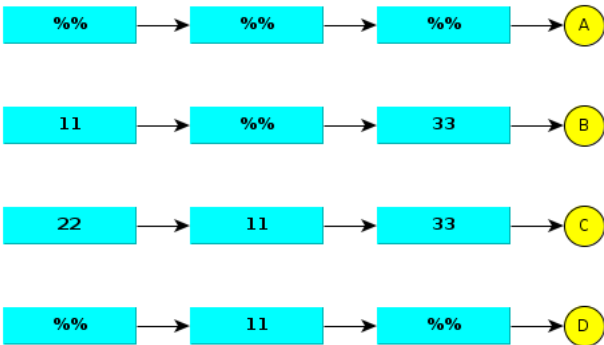
It also matched. So the value of n1 is now C.

This example exposed a case when the line order in the data file will differ from the processing order using the optimized data processing mode.

2.2 Linear mode

The linear mode was created to process values exactly in the order of the lines in the data file. No optimizations are done. Each line has its own path so matching is done for each column of each line.

Here is how the example from the optimized mode is represented in linear mode:



Now we receive a value with these properties:

	k1	k2	k3
Raw value 1	22	11	33

It will first match to A, then to C and finally to D. The output will be D.