VMware Smart Assurance Integration Guide

VMware Smart Assurance 10.0.0

vmware®

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2019 VMware, Inc. All rights reserved. Copyright and trademark information.

Contents

Preface 5

1 Event Log Processor 6
- Architectural Overview 6
- Configuration 7
- Rule Reference 9

2 Event Property Tagger 17
- Overview 17
- Configuration 18
- Usage 25
- Mutable and Immutable Events 34
- General Concepts on Matching Order 34

3 Data Collection Controller 40
- Overview 40
- DCC installation 40
- How authentication works 41
- DCC configuration 42
- Installing MongoDB 43
- Installing DC core 43
- Basic operations 44

4 Kafka Event Adapter 50
- Overview 50
- Installation 50
- Configuration 50
- Logging 55

5 Stream Collector 56
- Architecture 56

6 Smarts Collector 66
- Overview 66
- Configuration 66
- Logging 82

7 Smarts Listener 85

Overview 85

Installation 85

Configuration 86

Logging 91

Preface

As part of an effort to improve product lines, periodic revisions of software and hardware are released. Therefore, all versions of the software or hardware currently in use might not support some functions that are described in this document. The product release notes provide the most up-to-date information on product features. If a product does not function correctly or does not function as described in this document, contact a technical support professional.

Intended Audience

This document is part of the Smart Assurance UI documentation set and is intended for use by system administrators during the configuration of the Smart Assurance UI software.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation, go to http://www.vmware.com/support/pubs.

Event Log Processor

1

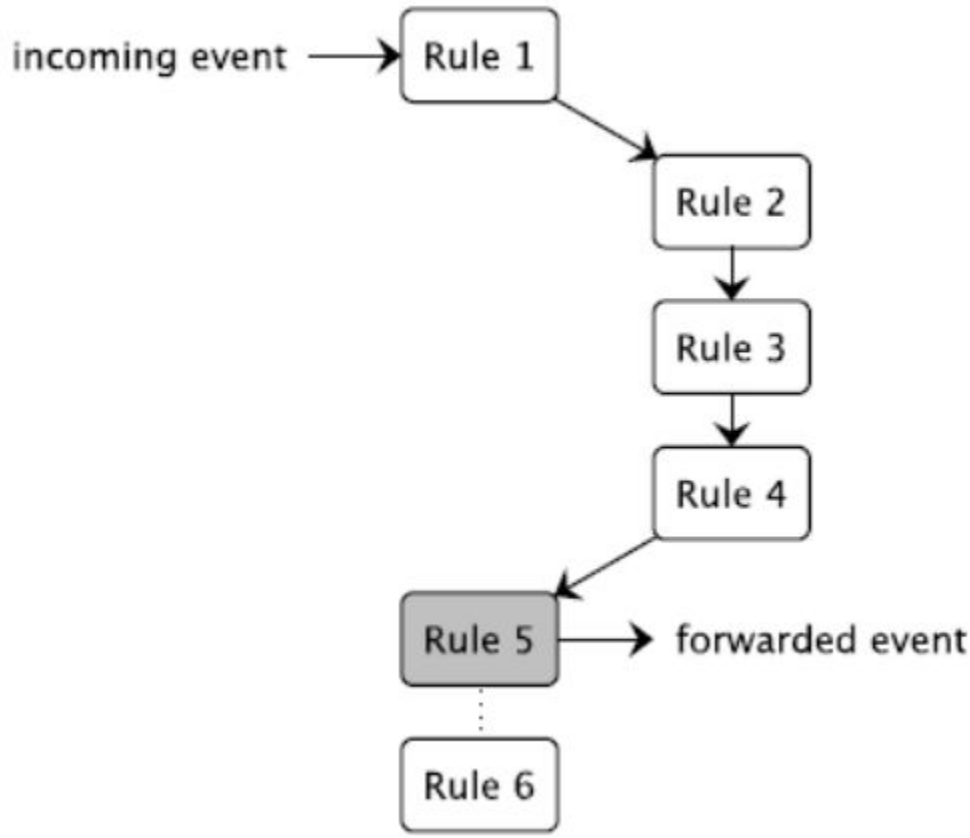
This chapter includes the following topics:

- Architectural Overview
- Configuration
- Rule Reference

Architectural Overview

The Event Log Processor processes events in a similar way a firewall processes packets. It evaluates a set of rules, which can be final (a final rule will be the last evaluated one) or not.

Figure 1-1. Rule Evaluation Example



In the *Rule Evaluation Example* figure, an event is submitted to the root rule (Rule 1) and gets propagated into the chain. Here Rule 5 is a final rule. An example of final rule is a rule that forwards the event to the next Processing Element. When the evaluation engine evaluates a rule, there are 4 possible results:

- continue : This is the default result of most rules. It simply tells the engine that evaluation should continue.
- success : A rule returns this result when it has successfully handled the incoming event. In firewall terminology, it very similar to an ACCEPT target. This is what happened on Rule 5 of the previous example.
- failure : This result is returned when a rule failed to evaluate. This typically means that the incoming event doesn't meet some preliminary condition that the rule is expecting. In firewall terminology, it very similar to a DROP target.
- error : This happens when a rule fails badly or unexpectedly for a reason which is not under its control. For example, a rule that would access an external resource could result in an error if the resource is not accessible.

Note Only the continue result will process further rules. The other results will stop processing within the current rule set and return control to the parent set. For each individual rule, you can override the result and replace it with another. For example, if a rule returns a failure because an event does not match a condition, you can choose to continue the evaluation by overriding the failure with continue . In the Rule Evaluation Example, the success result of Rule 5 could have been overridden with continue which would have forwarded the event to Rule 6 . This way it is possible to define complex backtracking nested chains that handle many different kinds of events and process them appropriately. See section Event Log Processor Configuration for an example of the override syntax.

Configuration

In order to use the Event Log Processor , you have to declare it in the Event Processing Manager configuration `< DCF-Install>/Event-Processing/Event-Processing-Manager/<INSTANCE ID>/conf/processing.xml`.

```
<processing-manager
    xmlns="http://www.watch4net.com/Events/DefaultProcessingManager"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.watch4net.com/Events/DefaultProcessingManager
DefaultProcessingManager.xsd ">

    <processing-element name="KAFKA" enabled="true"
        type="Kafka-Event-Adapter"
        config="Kafka-Event-Adapter/smarts-notifs-events/conf/kafka-event-adapter.xml" />
    <processing-element name="LOG-PROCESSOR"
        config="Event-Log-Processor/smarts-notifs-events/conf/event-log.xml"
        data="KAFKA" />
    <processing-element name="Smarts" enabled="true"
        config="Smarts-Listener/smarts-notifs-events/conf/smarts-listener.xml"
```

```
        data="LOG-PROCESSOR" />
    <processing-element name="EVENT-SPY" enabled="true"
        type="EventSpy" config="Event-Processing-Utils/smarts-notifs-events/conf" />
</processing-manager>
```

The above example declaration is from Smarts Events SP. The above configuration creates Event Log Processor and other components. The Event Log Processor accepts incoming events from Smarts listener. And, forward processed data to KAFKA which is another Processing Element through the data stream.

Event Log Processor Configuration

The Event Log Processor configuration basically consists of declaring the rules that we want incoming events to go through. As a simple example, the configuration for the Rule Evaluation Example can be written in the following way:

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://www.watch4net.com/Events/EventLogProcessor">
<rule name="Rule 1" />
<rules>
<rule name="Rule 2" />
<rule name="Rule 3" />
<rule name="Rule 4" />
</rules>
<rule name="Rule 5" />
<rule name="Rule 6" />
</rules>
```

The rules element allows for the creation of chains. A chain is evaluated as described in section 1 . This example uses a fictional < rule > tag to represent actual rules, which would be replaced with the syntax of the available rules as defined in section 3.

Each rule has an optional name attribute. It is mostly used as a description of the rule. It has no impact on evaluation but might help with troubleshooting since it is written in the log files in the event of an error.

To override the result of a rule, you can use the following attributes:

- on-continue to redefine the behavior of a rule that returns continue
- on-success to redefine the behavior of a rule that returns success
- on-failure to redefine the behavior of a rule that returns failure
- on-error to redefine the behavior of a rule that returns error

To continue the example from section 1 , a way to override Rule 5 could be the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<rules xmlns="http://www.watch4net.com/Events/EventLogProcessor">
<rule name="Rule 1" />
<rules>
<rule name="Rule 2" />
<rule name="Rule 3" />
<rule name="Rule 4" />
```

```
</rules>
<rule name="Rule 5" on-success="continue" />
<rule name="Rule 6" />
</rules>
```

A example implementation for Smarts events SP can be like below:

Sample file from sample implementation example: `<DCF-Install>/Event-Log-Processor/<INSTANCE ID>/conf/event-log.xml`

```
<rules xmlns="http://www.watch4net.com/Events/EventLogProcessor"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.watch4net.com/Events/EventLogProcessor EventLogProcessor.xsd">

       <format name="Unique ID" pattern="{0}" to="@Id" type="message">
           <field name="Id" type="LONG" />
       </format>

    <!-- Simply forward events to the data stream without modifying them. -->
    <forward stream="data" />

</rules>
```

In above configuration example the event set data is matched with "Id" as key, if any key having Id is found then it will get replaced with @Id. All other events which don't have this key will be automatically forwarded to next processing element.

Note To use below rules and formatting options of Event Log Processor needs similar kind of configurations in DCF installation.

Rule Reference

Basic Rules

Rule Chain (rules): The rule chain is the core of the Event Log Processor. It basically evaluates a list of rules as described in *Architectural Overview*. It does not require any specific configuration, but accepts name and override parameters as all rules do. A rule chain always forwards the result of the last evaluated rule.

```
<rules>
<rule name="Rule 1" />
<rule name="Rule 2" />
<rule name="Rule 3" />
</rules>
```

Include Rule (include): The include rule allows the administrator to break a large config file into multiple smaller files. This rule allows the use of patterns, which is a quick way to include many files. When a pattern selects multiple files, they are included in alphabetical order. The path to the included files is relative to the configuration file that references them.

```
<include name="Includes XML files in rules directory" dir="rules" includes="*.xml" />
```

Switch Rule (switch): The switch rule is a quick way to redirect the evaluation flow to a specific rule chain based on a field value. It tests the value of a field against different cases and evaluates the rules for the matching one. If a default is specified and the field value does not match any of the cases, the default rules are evaluated. By default, the comparison performed is strict , meaning that no type conversion is attempted (which means that 42 : 0 6 = 42 6 = "42"). However, with the strict attribute set to false , the rule will take care of converting the data before performing the comparison. The strict can be set separately on each case and/or globally for the whole switch.

```
<switch name="Switches based on Type field value" field="Type">
<case value="ERROR" type="STRING">
...
</case>
<default>
...
</default>
</switch>
```

Forward Rule (forward): The forward rule allows the propagation of events to other Processing Elements. The name of the output stream is defined by the stream attribute. Declaring this rule also allows the Event Processing Manager to link the given output stream name to another Processing Element . Any attempt to use an output stream that has not been declared in a forward rule will result in an error. The forward rule always returns success.

```
<forward name="Forwards events on the data stream" stream="data" />
```

Noop Rule (noop): The noop rule does nothing (no operation). However, its result is configurable, which allows its use as a DROP or ACCEPT rule. The rule result is configured with the result attribute.

```
<noop name="This rule does nothing" />
<noop name="This rule drops every incoming event" result="failure" />
```

Log Rule (log): The log rule prints events in a log file. This is mainly a troubleshooting rule that allows the administrator to understand what's going on during the evaluation. It always returns continue.

This rule uses the standard Java Logging API. The logger's name used by this rule can be configured with the logger attribute. If you modify your Event Processing Manager logging configuration (conf/logging.properties), this allows you, for example, to redirect logs from a specific log rule to different files. See *http:// download.oracle.com/javase/6/docs/technotes/guides/logging/overview.html* for more information.

```
<log name="Log these events" />
<log name="Log these events in another logger" logger="another.logger" />
```

Count Rule (count): The count rule is a specific instance of a log rule which periodically evaluates the current event rate. It uses the same logger attribute to change the logger and the period attribute to determine how often (in milliseconds) the event rate should be logged. This rule always returns continue.

```
<count name="Count these events and log every 5s" period="5000" />
```

Copy Rule (copy): The copy rule simply copies a field into another. This is useful to rename or reuse some fields. The field to be copied is defined with the field attribute and the destination field with the to attribute. Furthermore, the new copy can be converted in a specific output type with the type attribute. If this attribute is omitted, the original field type will be used. This rule returns continue if the field exists or else failure .

```
<copy name="Copy a to b" field="a" to="b" />
```

Set Rule (set): The set rule allows the definition of constant fields. It will set the value of the value attribute of the field defined in the to attribute. It optionally uses the datatype defined in type in order to not use the default string type. This rule always returns continue. If value is omitted, the resulting field will be null.

```
<set name="Sets a string field F containing V" to="F" type="STRING" value="V" />
<set name="Sets an integer field F containing 42" to="F" type="INT" value="42" />
```

Tagging Rules

Timestamp Rule (timestamp): The timestamp rule sets a field to the current time in milliseconds. The destination field is defined with the to attribute. To allow maximum performance, this rule accepts an accuracy attribute which defines how accurate the current time should be. For example, if accuracy is set to 0, the current time will be as accurate as possible, but if it is set to 1000, the current time will only be reliable up to the second. This rule always returns continue.

```
<timestamp name="Sets the current time in F" to="F" accuracy="100" />
<timestamp name="Sets the current time in F with the maximum accuracy" to="F" accuracy="
0" />
```

UUID Rule (uuid): The UUID rule sets a field to a newly random UUID (universal unique identifier). The destination field is defined with the to attribute. This rule always returns continue .

```
<uuid name="Generates a random UUID in F" to="F" />
```

Sequence Rule (seq): The sequence rule generates a sequence and sets the current value to the given field. The last attributed sequence number is not persisted, so you should not use it as an id. The destination field is defined with the to attribute. The initial sequence value can be set with the start attribute. This rule always returns continue .

```
<seq name="Generates sequence in F starting at 42" to="F" start="42" />
```

Id Generator Rule (id): The id generator rule is a combination of timestamp and sequence that allows the generation of a unique id (since the system clock is always increasing). It uses the same attributes used by the sequence rule and by the timestamp rule. This rule always returns continue.

```
<id name="Generates an id in Id" to="Id" />
```

Link Rule (link): The link rule is used to create a link between successive events. It is done by setting a field in each event that corresponds to the previous event id. The event id is set with the id attribute and the destination field with the to attribute. If the id is missing, the rule returns failure or else it returns continue .

```
<link name="Copies Id to pId in the next event" id="Id" to="pId" />
```

MAC Rule (mac): The MAC rule allow the creation of a Message Authentication Code for each event to protect their integrity. Combined with the link rule, this is a powerful tool to detect missing or altered events. A MAC rule uses a key and a list of fields to create the code. The code is added to the event as a binary field, whose name is defined by the to attribute. The source fields are set with the nested field elements. The MAC algorithm can be customized using the algorithm attribute. Each algorithm uses a key, which is specified using the key attribute. The key attribute format is base64:<base 64 representation of the key> and can optionally be encrypted using DCF crypt-password command.

```
<mac name="Creates a MAC of the Id,pId,Message of each events" key="base64:a2V5Cg==" to=
"mac" algorithm="HMac-MD5">
<field name="Id" type="LONG" />
<field name="pId" type="LONG" />
<field name="Message" type="STRING" encoding="ASCII" />
</mac>
```

Format Rule (format): The format rule allows the combination of multiple fields to create a new one using a format string. The destination field is set with the to attribute and the format string with the pattern attribute. The rule support two types of format string (set with the type attribute):

- **message** is the default format and uses a Java MessageFormat . For more information on the format string syntax, refer *http://download.oracle. com/javase/6/docs/api/java/text/MessageFormat.html*.

- **printf** uses a Java Formatter which format string syntax is similar the the wellknown C printf . For more information on the format string syntax, refer *http://download.oracle.com/javase/6/docs/api/java/util/Formatter.html*. The format string usually contains placeholders to insert some field values (for example, %s). Each placeholders must be attribute its value with a nested field element. By default, each field is handled as a string , however, some placeholders might expect different data types (for example, %d would expect an integer). You can force a type conversion with the type attribute (one of INT, SHORT, FLOAT, DOUBLE, LONG, NUMERIC, STRING, BOOLEAN, BYTE, BYTES or OBJECT is expected).

```
<format name="Prints Id=Type into IdType" pattern="{0}={1}" to="IdType" type="message">
<field name="Id" type="LONG" />
<field name="Type" type="STRING" />
</format>
```

Matching Rules

Exists Rule (exists): This rule returns continue if a specific field (designated by the field attribute) exists. If not, it returns failure . The allow-null attribute determines the behavior if the field exists but is null. The default is false which treats null as if the field does not exist.

```
<exists name="Match messages with a field f" field="f" allow-null="false" />
```

Starts With Rule (starts-with): This rule returns continue if a specific field (designated by the field attribute) start with the value defined in the prefix attribute. If not, it returns failure.

```
<starts-with name="Match messages starting with ERROR" field="Message" prefix="ERROR" />
```

Contains Rule (contains): This rule returns continue if a specific field (designated by the field attribute) contains the value defined in the value attribute. If not, it returns failure.

```
<contains name="Match messages containing ERROR" field="Message" value="ERROR" />
```

Equals Rule (equals): This rule returns continue if a specific field (designated by the field attribute) is equal to the value defined in the value attribute. If not, it returns failure. By default, the comparison performed is strict , meaning that no type conversion is attempted (which means that 42 : 0 6 = 42 6 = "42"). However, with the strict attribute set to false , the rule will take care of converting the data before performing the comparison.

```
<equals name="Match messages which type is ERROR" field="type" value="ERROR" type="
STRING" />
<equals name="Match severities of 0, 0.0, ..." field="severity" value="0" type="INT"
strict="false" />
```

Not Equals Rule (not-equals): This rule is the exact opposite of the Equals Rule (equals) . It returns failure if a specific field (designated by the field attribute) is equal to the value defined in the value attribute. If not, it returns continue . This rule also support the strict attribute of the Equals Rule (equals).

```
<not-equals name="Match messages which type is not DEBUG" field="type" value="DEBUG"
type="STRING" />
```

Regex Rule (regex): The regex rule has two main purposes: match a field value against a regular expression and optionally extract some data from it. If the regular expression given in the pattern is found in the field designated by the field attribute, this rule will try to extract some data from the value and return continue . If the value doesn't match, the rule will return failure . The data extraction is configured with a nested extraction element whose name attribute is the destination field and value attribute the extraction string. The extraction string can refer to the group matched in the field value using the $ <ref> syntax. If it is missing, a tail extraction is performed, meaning that everything after the match is extracted. For complete documentation of the supported regular expression dialect, refer the *Java documentation*.

```
<regex name="Extracts a type from a message" field="Message" pattern="^(ERROR|INFO): ">
<extraction name="Type" value="$1" />
<extraction name="Message" />
</regex>
<regex name="Match messages starting with ERROR" field="Message" pattern="^ERROR: ">
```

Parsing Rules

Date Parser Rule (parse-date): This rule converts the string representation of a date into a timestamp in milliseconds. The date format can be configured using the pattern and locale attributes. When using the default parser, the pattern must be compatible with the standard Java data parser. When the type attribute is set to rfc3164 , the pattern is not required and the parser will interpret RFC 3164 TIMESTAMP type (which imply guessing the year as it is not included in a RFC 3164 TIMESTAMP). The locale must be a RFC 3066 compliant locale id. It is used to decode some locale dependent elements of the date (e.g. the month name). Optionally, a time-zone can be specified with the timezone attribute. This is the time-zone of the date to parse when it is not specified in the date itself. The field to parse is designated by the attribute field and the destination field by the attribute to . If to is omitted, field is used (this overwrites the previous value). This rule returns continue if the date field has been successfully parsed, failure if not.

```
<parse-date name="Parses a date" field="Date" to="Timestamp" pattern="MMM dd yyyy
HH:mm:ss" />
<parse-date name="Parses a date in GMT" field="Date" to="Timestamp" timezone="GMT"
pattern="MMM dd yyyy HH:mm:ss" />
```

Duration Parser Rule (parse-duration): This rule converts a duration in the format [[hh:]mm:]ss into a number of milliseconds. The field to parse is designated by the attribute field and the destination field by the attribute to . If to is omitted, field is used (this overwrites the previous value). The separator can be changed with the attribute separator . This rule returns con- tinue if the duration field has been successfully parsed, failure if not.

```
<parse-duration name="Parses a duration" field="Duration" to="DurationMsec" />
```

Syslog Priority Parser Rule (parse-pri): This rule converts a Syslog Priority field of a syslog record into its corresponding Severity and Facility as described in the RFC 5424 . The field to parse is designated by the attribute field and the destination field by the attributes facility and severity. This rule returns continue if the Priority field has been successfully parsed, failure if not.

```
<parse-pri name="Parses a Syslog Priority" field="Pri" facility="Facility" severity="
Severity" />
```

IPv4 To Integer Rule (ipv4-to-int): This rule converts the string representation of an IP address into a compact 4 byte representation. This is especially useful for IP addresses stored in a database, as it allows more compact data and faster queries. The field to parse is designated by the attribute field and the destination field by the attribute to . If to is omitted, field is used (this overwrites the previous value). This rule returns continue if the IP address has been successfully converted, failure if not.

```
<ipv4-to-int name="Parses an IPv4" field="Address" />
```

Integer To IPv4 Rule (int-to-ipv4): This rule converts the integer representation of an IP address into a standard dot notation. This is especially useful for IP addresses coming from a Flow Listener. The field to parse is designated by the attribute field and the destination field by the attribute to . If to is omited, field is used (this overwrites the previous value). This rule returns continue if the IP address has been successfully converted, failure if not.

```
<int-to-ipv4 name="Integer to IPv4" field="Address" />
```

Advanced Rules

Tokenizer Rule (tokenize): The tokenize rule generates token events by splitting a field into tokens. This rule can be used to create a token stream in order to build a keyword index and make the search more user-friendly. The tokenized field can be set with the field attribute. and the tokenizer type with the type attribute. Currently, the tokenize rule supports two tokenizer types:

- **simple** uses some stop characters (set with the pattern attribute) to split the field into tokens.

- **lucene** uses the well known Lucene tokenizer components to split the message. The underlying Lucene tokenizer is designated by the class attribute. A tokenize rule acts as a rule chain. Token events generated by the tokenizer rule will be submitted to the nested rule chain. A typical use case is to forward them to a specific output stream in order to process them separately like in the following examples (note the on-success="continue" to make sure that all tokens are forwarded and that evaluation continues after the tokenize rule):

```
<tokenize field="Message" to="Token" type="simple" pattern="[ \r\n\t]">
<forward name="Forwards tokens on the token stream" on-success="continue" stream="
token" />
</tokenize>
<tokenize field="Message" to="Token" type="lucene" class="org.apache.lucene.analysis.
SimpleAnalyzer">
<forward name="Forwards tokens on the token stream" on-success="continue" stream="
token" />
</tokenize>
```

Custom Rule (custom): The custom rule is an extension point for Java developers that would like to create their own rules. The only required attribute is class pointing to the custom rule class name. Every other attribute or element is private to the custom rule. A custom rule must extend the com.watch4net.events.processing.processor.rules.CustomRule class from event-log-processor.jar to be recognized by the Event Log Processor.

```
<custom name="A custom rule" class="com.company.MyRule" setting="value">
<parameter>value</parameter>
</custom>
```

Event Property Tagger

2

This section provides information about the Event Property Tagger and the way to use it and includes all necessary information to configure and run the Event Property Tagger. Administrator can add new properties to events based on the properties these events have. The Event Property Tagger can only be used within an instance of the Event Processing Manager version 5.0 and later are supported.

This chapter includes the following topics:

- Overview
- Configuration
- Usage
- Mutable and Immutable Events
- General Concepts on Matching Order

Overview

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

Configuration Overview

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

1. Refer Features section 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).
4. Configure the Event Property Tagger configuration file (configure additional instances of the file if using multiple Event Property Tagger's).
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.

6 Restart the Event Processing Manager.

Note 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.

Features

- **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.
- **New properties based on same ”key”**: A "key" can be used to add more than one new property.
- **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.
- **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.
- **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. DCF can compensate for this behavior as explained in Mutable and Immutable Events section. 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.

- **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.

Configuration

Event Processing Manager configuration

This module must be declared in the Event-Processing-Manager `<DCF Install>/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.

Event Property Tagger configuration

Configuration file: The event-property-tagger.xml configuration file is needed to run the module. It is located at: `<DCF Install>/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">
<!-- 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" decoder="IP">
src_addr</key-property>
<key-property delete-after-use="false" string-type="regex" decoder="INT_NATIVE"
parser="INT_NATIVE">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>
<!-- 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 reenable 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 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, 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 section.

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 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 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 sqlpattern s, regex es 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 parser attribute is optional and defines how the configuration csv value will be loaded into the tagging engine. This allows direct match to be faster for some case where you don't need to convert the incoming data but only the configuration data and decode the incoming data into it's native representation. The current implementations are the following:

- **BYTES**: This will retrieve a field value from the event through getBytes().
- **INT NATIVE**: This will retrieve a field value from the event through getInt().
- **INT**: This will retrieve a field value from the event through getString() and then convert it to int.
- **IP**: This retrieve the bytes from the event through getBytes() call and will create either an IPv4 or IPv6 string representation depending on the number of bytes associated to that field.
- **STRING**: Default implementation which returns a String object for comparison 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 be have the suffix 0 and 1.

Input file: The input file uses a "chosen string" as a seperator 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. Refer the configuration file, 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.

Accessors

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.

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.

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.

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.

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.

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, the Event Property Tagger runs with the Event-Processing-Manager.

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>
```

Wrong example of an input file:

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

This wrong 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*.

Correct example of an input file:

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

Using the correct 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

Return 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

According to above example, it is possible to add ernest for the admin property because it overwrites the david entry.

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 fl7- and whose devtype is host.

The value 8thFloor-All is needed for all events whose device property begins with fl8- 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:

```
fl7 -.*, host, 7thFloor-Hosts
fl8 -.*, .*, 8thFloor-All
.* r[0-9]{3}.*, router, Routers-R
***, ***, Others
```

Using the example above and processing the following events:

	device	devtype
Event 1	fl7-asd-56	host
Event 2	fl7-fas-42	router
Event 3	fl8-rew-rwe	host
Event 4	fl123-qwe-r342-wq	router

Returns these results:

	device	devtype	group
Event 1	fl7-asd-56	host	7thFlppr-Hosts
Event 2	fl7-fas-42	router	Others
Event 3	fl8-rew-rwe	host	8thFloor-All
Event 4	fl123-qwe-r342-wq	router	Router-R

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

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 section.

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.

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

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

Mutable and Immutable Events

Immultable 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 DCF 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 DCF (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.

General Concepts on Matching Order

Optimized mode (legacy)

As illustrated in the examples above, raw value's properties are matched against the input files line-byline. 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.

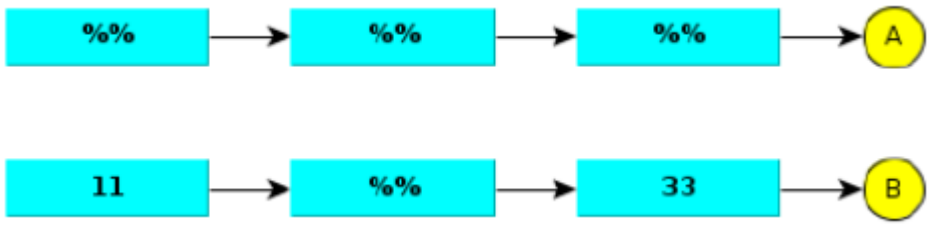
```
%%,%%,%%,A
```

Will be represented internally in a tree like that:

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

Then the second line:

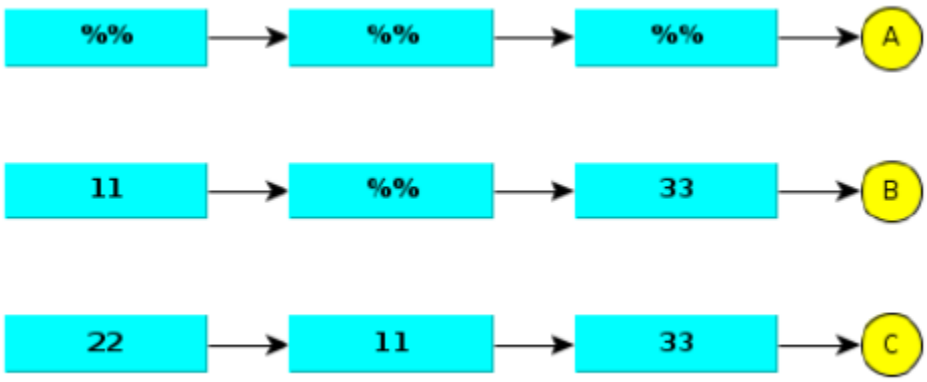
```
11,%%,33,B
```



A new path will be created.

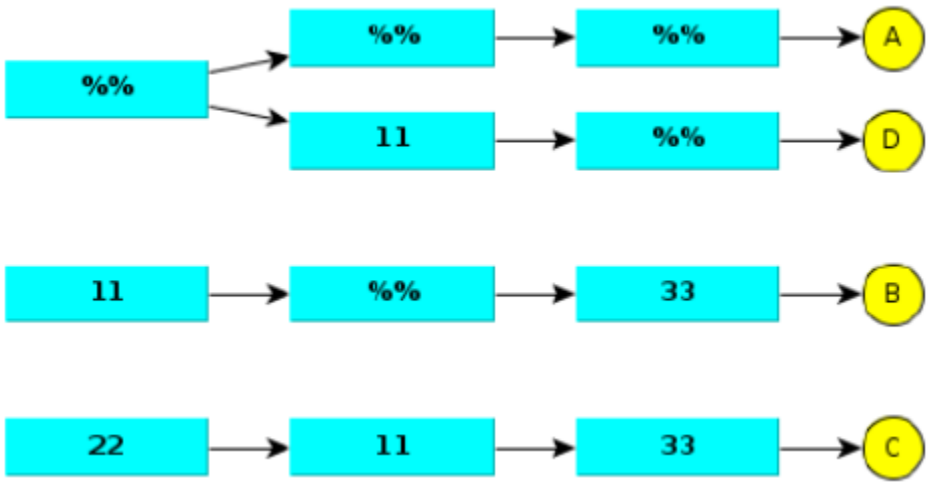
Same for third line:

```
22,11,33,C
```



And then the fourth line:

```
%%,11,%%,D
```



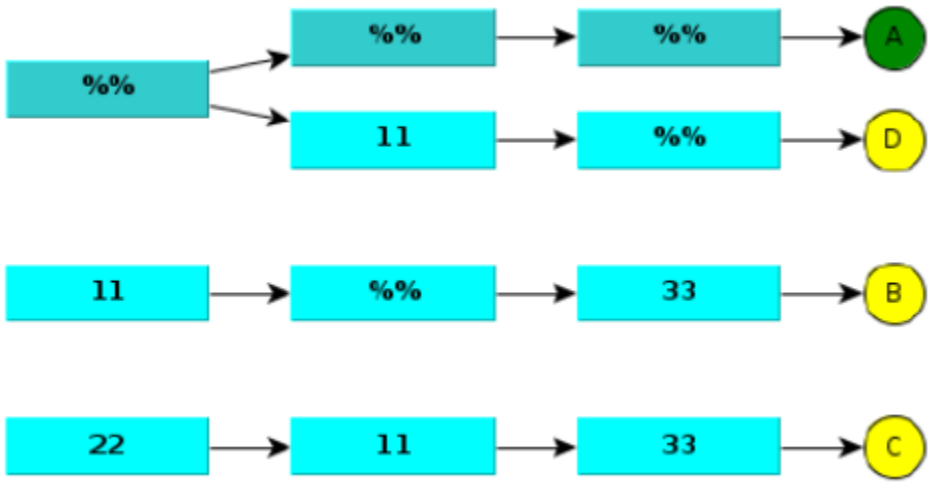
Here is the catch; because the fourth line first column shares the same value as the first line first column, its path is shared with the first line. That way the common expression will only be checked once.

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:

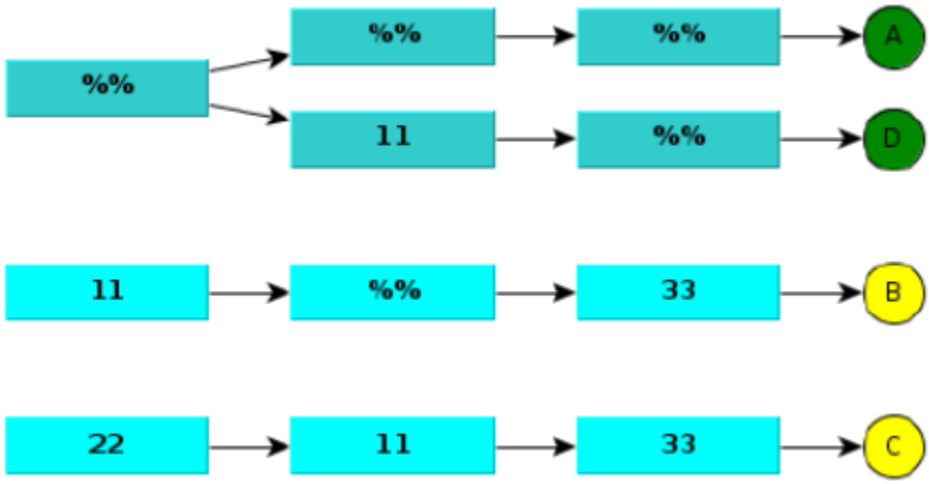
	k1	k2	k3
Raw value 1	22	11	33

Tree paths are followed step by step:



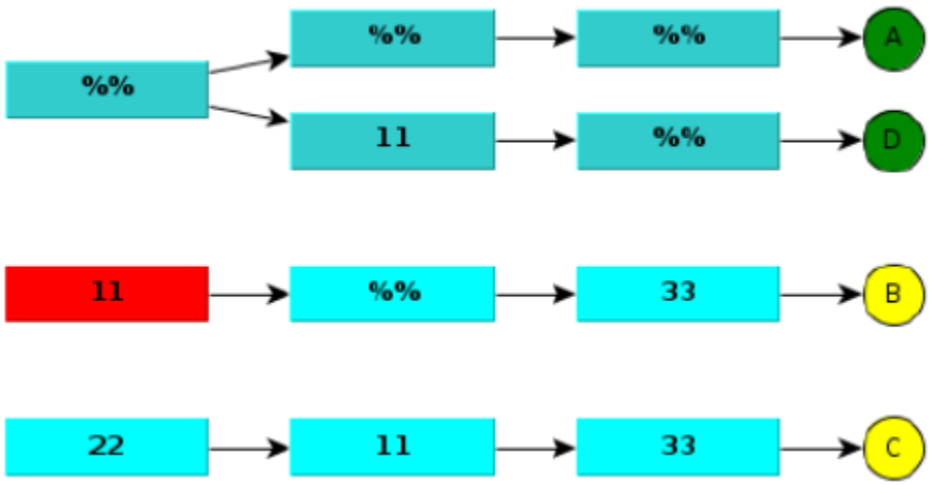
The first path matched so the new property n1 is created and assigned value A.

Then the second path:



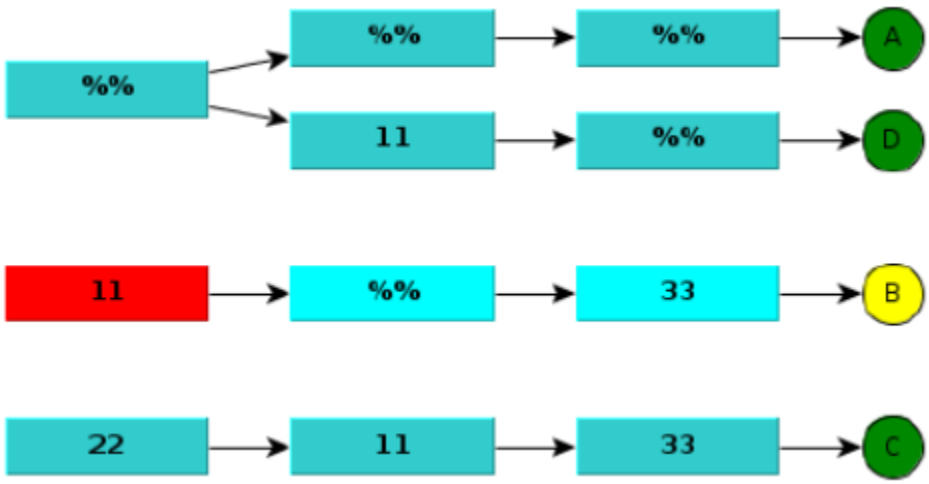
It also matched. So the value of n1 is now D.

Then the third path:



Not matched.

Finally the fourth path:



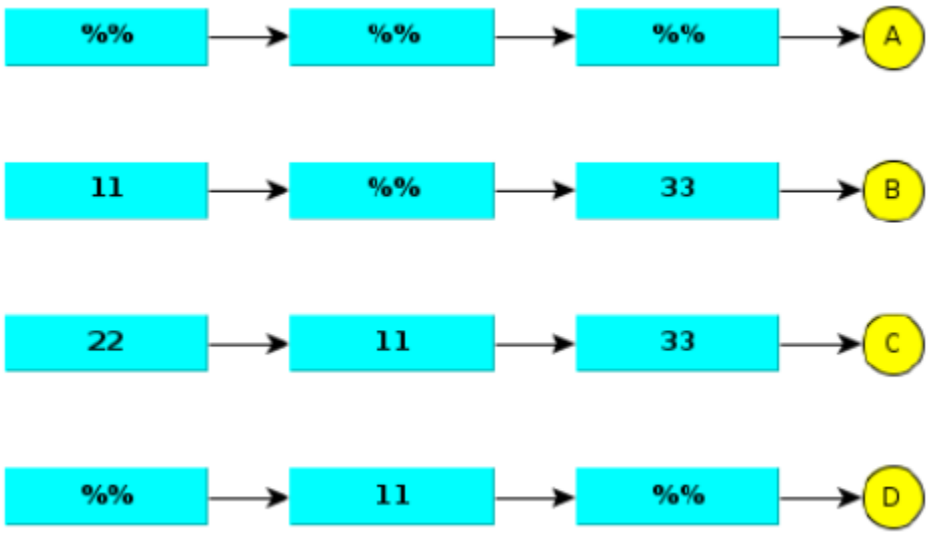
It also matched. So the value of n1 is now C.

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.

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

Data Collection Controller

3

This chapter includes the following topics:

- Overview
- DCC installation
- How authentication works
- DCC configuration
- Installing MongoDB
- Installing DC core
- Basic operations

Overview

Data Collection Controller (DCC) is a REST-based orchestrator that takes requests from users to deploy and manage data collection blocks. Data Collection Blocks (usually called 'block') collect data from specified devices, transform, filter, and publish them via multiple interfaces.

Currently, a block defines a pipeline that may contain one or more components that will collect, process, and publish data.

DCC installation

This section explains, how to install DCC component.

Procedure

1 Download the package and give it 'run' access:

```
chmod +X dcf-installer.sh
       sh dcf-installer.sh
```

2 Accept EULA and configure parameters:

```
- Do you accept EULA [y/N] ?
    - Install in /opt/DCF [Y/n] ?
    - Required dependencies

       [1]   java '8.0.141' v8.0.141\
       [2]   module-manager '1.13u1' v1.13u1\
       [3]   webservice-gateway 'Default' (none) => v2.3u2\
       [4]   python 'Default' (none) => v3.6.5\
       [5]   controller 'Default' (none) => v1.0\Certificate generation

     * Finalizing installation...
     * Installing service 'webservice-gateway Default'...              [ installed ]
     * Generating server certificate...                                     [ done ]
     * Generating client certificate...                                     [ done ]
     * Generating truststore...                                             [ done ]
     * Generating trusted certificate...                                    [ done ]
```

3 Complete installation and start services

```
    Installation complete.

     * Starting 'webservice-gateway Default'...                              [ OK ]
     * Starting 'task-scheduler Default'...                                  [ OK ]
     * Starting 'controller Default'...                                      [ OK ]
    ```

* IMPORTANT: All useful utilities are located in "/opt/DCF/bin".
To install additional modules, use "/opt/DCF/bin/manage-modules.sh"

* WARNING: Anti-virus software are known to severely impact VMware Service Assurance DCF
performance and reliability. It is recommended to add
"/opt/DCF" to your anti-virus white-list. Please contact your vendor for more information about
white-listing.

* Installing packages\
New modules (blocks) are distributed in a packaged format, usually using the .pkg extension.
Once copied in the Module-Respository directory, DCC is able to detect changes and make them
available for provisioning.
```

# How authentication works

DCC is configured to be the only entry point of DCF. Once you install it, it allows you to provision and orchestrate any available block in DCF.

Most of the DCC configuration is done through properties in `dc_controller/server/config.py`.

This is typically where an administrator would change the protocol (http/https), the listening address, the listening port, secret key parameters, and a few other advanced settings.
```

DCC communication is encrypted using HTTPS. The client issues a request using Basic authentication through a HTTPS channel. Therefore, a certificate needs to be generated during the installation process.

Manually generating a certificate

To manually generate certificate, admin needs to have OpenSSL.

```
openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days 365
```

DCC configuration

The configuration file `dc_controller/server/config.py` contains all the parameters of DCC. It contains information about DCC internal credentials, secret key, path to client certificate. MongoDB location and credentials, DC Core location and credentials, maximum number of clients interacting with web service gateway, DCC location, and type of deployment (i.e. Bare metal or virtualized). This file will be generated as result of the 'create first user'.

```
MONGODB_URI='127.0.0.1:27017'
MONGODB_USERNAME='admin'
MONGODB_PASSWORD='ENCRYPTED_PASSWORD'
MONGODB_DATABASE='admin'
DEPLOYMENT_TYPE='BAREMETAL'
MNR_HOST='127.0.0.1'
MNR_PORT='48443'
MNR_USER='admin'
MNR_PASSWORD='ENCRYPTED_PASSWORD'
MNR_MAX_THREADS=2
SECRET_KEY='SECRET_KEY'
CERTIFICATE_PATH='/opt/DCF/Tools/Controller/Default'
DCF_HOST='0.0.0.0'
DCF_PORT='8443'
USERNAME='admin'
PASSWORD='ENCRYPTED_PASSWORD'
LOG_LEVEL='INFO' # 'DEBUG', 'ERROR', 'CRITICAL', 'WARNING'

# The following properties are passed to the DCC web server (Gunicorn)
bind='0.0.0.0:8443' # Gunicorn bind socket
workers='4' # Number of async workers
worker_class='gthread' # Gunicorn mode
certfile='/opt/DCF/Tools/Controller/Default/conf/controller-cert.pem'
keyfile='/opt/DCF/Tools/Controller/Default/conf/controller-key.pem'
preload_app='True'

# The following is a Gunicorn server hook, called just after the server is started.
def when_ready(server):
    import dc_controller.wsgi as dcfControllerApp
    dcfControllerApp.create_default_template()
```

Installing MongoDB

This section explains, how to install the MongoDB.

1 Edit the file `/etc/yum.repos.d/mongodb-org-4.0.repo` with the following content:

```
[mongodb-org-4.0]
    name=MongoDB Repository
    baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.0/x86_64/
    gpgcheck=1
    enabled=1
    gpgkey=https://www.mongodb.org/static/pgp/server-4.0.asc
```

2 Install MongoDB, by invoking the command:

```
yum install -y mongodb-org
```

3 Create first user in MongoDB, by invoking the command:

```
# Enter into Mongo command-line Shell
    mongo
    # Select admin database inside Mongo shell
    > use admin;
    # Create first user
    > db.createUser(
    ...   {
    ...     user: "admin",
    ...     pwd: "changeme",
    ...     roles: [ { role: "root", db: "admin" } ]
    ...   }
    ... )
```

4 Enable authentication in MongoDB, by invoking the command:

```
vi /etc/mongod.conf
    security:
    authorization: "enabled"
```

5 Restart MongoDB, by invoking command:

service mongod restart

Installing DC core

This section explains, how to install the DC core.

Procedure

1 Download the DC core, from support.vmware.com.

2 Install DC core, by invoking the command:

```
chmod +X release-1.0.0.39.sh && sh release-1.0.0.39.sh
```

3 Goto the directory:

```
/opt/DCF/bin/manage-module.sh
```

4 Start the Controller, by invoking the command:

```
service start controller
```

Basic operations

The following are the general steps a typical client would complete after installing DCC to access the REST API using curl. Samples and examples in this section provide working illustrations of these concepts.

Setup a new collector:

1 Create block:

```
Request : curl --user admin:changeme --cacert /opt/DCF/Tools/Controller/Default/cert.pem -X POST -
H 'Content-Type: application/json' https://127.0.0.1:8443/dcc/v1/catalog/blocks/smarts-
metrics/provision

    Response :
        {
          "blockID": "smarts-metrics",
          "message": "Block successfully deployed",
          "time": "2019-02-25 23:25:06.824015",
          "instanceId": "smarts-metricseaea8c69-b12a-4fa6-8389-6ea87dd997c5",
          "params": {
            "smarts-metrics": {
              "primary": {
                "collect": {
                  "sm": [
                    {
                      "host": "localhost",
                      "port": "12345",
                      "name": "INCHARGE-AM-PM",
                      "username": "admin",
                      "password":
"{FAF745B9C01D858B45D7F644CE55D30064E409A847AA2351B30B6FB74B3BA7FDEC5A9315ECA00C2F69D9C425039CA1F1}
",
                      "type": "both",
                      "topologyPollingPeriod": "3600",
                      "metricPollingPeriod": "240",
                      "backendgroup": "group",
                      "thread": {
                        "poolsize": "3"
                      },
```

```
                      "dmconnection": {
                        "poolsize": "3"
                      }
                    }
                  ]
                }
              },
              "kafka": {
                "cluster": {
                  "node": [
                    {
                      "host": "localhost",
                      "port": "9092"
                    }
                  ],
                  "topic": "default-topic",
                  "auth": "true",
                  "username": "admin",
                  "password": "admin-secret"
                }
              }
            }
          }
        }
```

2 Configure block:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem --user admin:changeme --request
POST --url https://127.0.0.1:8443/dcc/v1/runtime/blocks/smarts-metrics70d9afac-516a-47f5-
b38f-799fa530d55c/config/set --header 'content-type: application/json' --data '{
      "smarts-metrics": {
        "primary": {
          "collect": {
            "sm": [
              {
                "host": "localhost",
                "port": "12345",
                "name": "INCHARGE-AM-PM",
                "username": "admin",
                "password":
"{544756F82CDB79957BC6004EA85A6F97CA541AE9AB6CDF4F8D1A99BD2D341B6AF8A75F35A5DE5C59651AFF574DBC604E}
",
                "type": "both",
                "topologyPollingPeriod": "3600",
                "metricPollingPeriod": "240",
                "backendgroup": "group",
                "thread": {
                  "poolsize": "3"
                },
                "dmconnection": {
                  "poolsize": "3"
                }
              }
            ]
```

```
          }
        },
        "kafka": {
          "cluster": {
            "node": [
              {
                "host": "localhost11",
                "port": "9092"
              }
            ],
            "topic": "default-topic",
            "authType": "SASL_PLAINTEXT",
            "username": "admin",
            "password":
"{28811D69F2FEEA51A491E8DF763912A5B692517875494C8B524D712CF3576658B1E0ED1CF87F00B0F88E4EFAA487A5D6}
"
          }
        }
      }
    }'

    Response : New configuration successfully set in block: smarts-metricseaea8c69-
b12a-4fa6-8389-6ea87dd997c5  for smarts-metrics service
```

3 Start block:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem --user admin:changeme --request
POST --url https://127.0.0.1:8443/dcc/v1/runtime/blocks/smarts-metrics70d9afac-516a-47f5-
b38f-799fa530d55c/service/start

    Response : [{"id": "apg-collector-manager-smarts-metricseaea8c69-b12a-4fa6-8389-6ea87dd997c5",
"module": "collector-manager", "instance": "smarts-metricseaea8c69-b12a-4fa6-8389-6ea87dd997c5",
"status": "STARTED", "last-modified": "1551155145000", "startup-mode": "auto", "start-order":
"96", "stop-order": "4"}]
```

4 Stop block:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem --user admin:changeme --request
POST --url https://127.0.0.1:8443/dcc/v1/runtime/blocks/smarts-metrics70d9afac-516a-47f5-
b38f-799fa530d55c/service/stop

    Response : [{"id": "apg-collector-manager-smarts-metricseaea8c69-b12a-4fa6-8389-6ea87dd997c5",
"module": "collector-manager", "instance": "smarts-metricseaea8c69-b12a-4fa6-8389-6ea87dd997c5",
"status": "STOPPED", "last-modified": "1551155147000", "startup-mode": "auto", "start-order":
"96", "stop-order": "4"}]
```

5 Restart Service:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem --user admin:changeme --request
POST --url https://127.0.0.1:8443/dcc/v1/runtime/blocks/smarts-metrics70d9afac-516a-47f5-
b38f-799fa530d55c/service/restart
```

```
    Response : [{"id": "apg-collector-manager-cloudify-
orchestratorffa3bb75-99f4-4613-8a5a-45e2cee0372c", "module": "collector-manager", "instance":
"cloudify-orchestratorffa3bb75-99f4-4613-8a5a-45e2cee0372c", "status": "STARTED", "last-modified":
"1551155175000", "startup-mode": "auto", "start-order": "96", "stop-order": "4"}]
```

6 Get block object:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem -u admin:changeme --request
GET   --url https://127.0.0.1:8443/dcc/v1/catalog/blocks/smarts-metricsAPI/availability

    Response : "smarts-metrics"
```

7 Get instantiated block details:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem -u admin:changeme --request
GET   --url https://127.0.0.1:8443/dcc/v1/catalog/blocks/instances

    Response : {"smarts-metrics4cc747e0-da89-433e-9233-a6895b89755b": {"block_object_id": "smarts-
metrics", "deployment_status": "Running", "dc_service_status": "STOPPED", "poll_count": 16},
"smarts-metrics035f8121-aaed-4d66-83eb-ae3a7841f720": {"block_object_id": "smarts-metrics",
"deployment_status": "Running", "dc_service_status": "STOPPED", "poll_count": 16}}
```

8 Get block config:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem --user admin:changeme --request
GET --url https://127.0.0.1:8443/dcc/v1/runtime/blocks/smarts-metrics70d9afac-516a-47f5-
b38f-799fa530d55c/config/get

    Response :
        {
          "smarts-metrics": {
            "primary": {
              "collect": {
                "sm": [
                  {
                    "host": "localhost",
                    "port": "12345",
                    "name": "INCHARGE-AM-PM",
                    "username": "admin",
                    "password":
"{FAF745B9C01D858B45D7F644CE55D30064E409A847AA2351B30B6FB74B3BA7FDEC5A9315ECA00C2F69D9C425039CA1F1}
",
                    "type": "both",
                    "topologyPollingPeriod": "3600",
                    "metricPollingPeriod": "240",
                    "backendgroup": "group",
                    "thread": {
                      "poolsize": "3"
                    },
                    "dmconnection": {
                      "poolsize": "3"
                    }
                  }
```

```
                ]
              }
            },
            "kafka": {
              "cluster": {
                "node": [
                  {
                    "host": "localhost",
                    "port": "9092"
                  }
                ],
                "topic": "default-topic",
                "auth": "true",
                "username": "admin",
                "password": "admin-secret"
              }
            }
          }
        }
```

9 Get Packages:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem --user admin:changeme --request
GET --url https://127.0.0.1:8443/dcc/v1/catalog/blocks/packages

    Response : ["clearwater", "cloudify-orchestrator", "smarts-metrics", "smarts-notifs",
"velocloud-sdwan", "vrops"]
```

10 Get Services

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem --user admin:changeme --request
GET --url https://127.0.0.1:8443/dcc/v1/catalog/blocks/smarts-metrics70d9afac-516a-47f5-
b38f-799fa530d55c/services

    Response : {"id": "apg-collector-manager-smarts-metricseaea8c69-b12a-4fa6-8389-6ea87dd997c5",
"module": "collector-manager", "instance": "smarts-metricseaea8c69-b12a-4fa6-8389-6ea87dd997c5",
"status": "STOPPED", "last-modified": "-1", "startup-mode": "auto", "start-order": "96", "stop-
order": "4"}
```

11 Get Default Block Config:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem -u admin:changeme --request
GET   --url https://127.0.0.1:8443/dcc/v1/catalog/blocks/smarts-metrics/config/default

    Response :
        {
          "smarts-metrics": {
            "primary": {
              "collect": {
                "sm": [
                  {
                    "host": "localhost",
                    "port": "12345",
```

```
                    "name": "INCHARGE-AM-PM",
                    "username": "admin",
                    "password":
"{CCC8B6EFC8D913616264F4B8A0D25B5A52DBFAFE4F6EB5BB94B86D4B25BC2FA7F11E59A9A733BBA10002A20C5D60EE6A}
",
                    "type": "both",
                    "topologyPollingPeriod": "3600",
                    "metricPollingPeriod": "240",
                    "backendgroup": "group",
                    "thread": {
                      "poolsize": "3"
                    },
                    "dmconnection": {
                      "poolsize": "3"
                    }
                  }
                ]
              }
            },
            "kafka": {
              "cluster": {
                "node": [
                  {
                    "host": "localhost",
                    "port": "9092"
                  }
                ],
                "topic": "default-topic",
                "authType": "SASL_PLAINTEXT",
                "username": "admin",
                "password":
"{B07EA85D8D516D4CA4201CFC8DEBEE022E6DE44C0AE6CB9652FC850F08859AFF5A93A33A85220F3E9E1719D3595D6AA7}
"
              }
            }
          }
        }
```

12 Delete block:

```
Request : curl --cacert /opt/DCF/Tools/Controller/Default/cert.pem -u admin:changeme --request
DELETE   --url https://127.0.0.1:8443/dcc/v1/catalog/blocks/smarts-metrics035f8121-aaed-4d66-83eb-
ae3a7841f720

    Response : {"instanceId": "smarts-metrics035f8121-aaed-4d66-83eb-ae3a7841f720", "message":
"Block instance successfully deleted", "time": "2019-04-09 05:42:21.204886"}
```

Kafka Event Adapter

4

This chapter includes the following topics:

- Overview
- Installation
- Configuration
- Logging

Overview

The Kafka Event Adapter is a component that is able to both read events from kafka and write events to kafka.

DCF management packs takes care of installing and configuring related Kafka event modules. So it is assumed that Kafka and the Kafka Event Adapter have been installed properly.

Installation

The installation of the Kafka Event Adapter happens through DCF controller. The Kafka Event Adapter is provided as a module available in the modules repository. The module is installed and controlled like all other DCF modules/services.

Below command gives all the kafka event modules info:

```
<DCF-Install>/bin/manage-modules.sh  list available | egrep kafka-event

 * kafka-event-adapter-1.0u1-linux-x64               : Event-Processing Kafka-Event-
Adapter                 v1.0u1   linux-x64 installed (2)
```

Configuration

Event-Processing-Manager configuration

As part of management pack installation through controller, Kafka Event Adapter module will be installed and also its dependency modules like the Event-Processing-Manager will also be installed.

In below example, anything that the smarts-listener receives will be forwarded to the kafka-event- adapter for the adapter to write to Kafka.

File: `<DCF-Install>/Event-Processing/Event-Processing-Manager/<INSTANCE>(ex: smarts-notifs-events)/conf/processing.xml`

```
<processing-manager xmlns="http://www.watch4net.com/Events/DefaultProcessingManager"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.watch4net.com/Events/DefaultProcessingManager
DefaultProcessingManager.xsd ">
        <processing-element name="KAFKA" enabled="true" type="Kafka-Event-Adapter" config="Kafka-Event-
Adapter/smarts-notifs-events/conf/kafka-event-adapter.xml"/>
        <processing-element name="Smarts" enabled="true" config="Smarts-Listener/smarts-notifs-
events/conf/smarts-listener.xml" data="KAFKA"/>
        <processing-element name="EVENT-SPY" enabled="true" type="EventSpy" config="Event-Processing-
Utils/smarts-notifs-events/conf"/>
</processing-manager>
```

KAFKA processing element deals with publishing the event data from Smarts Processing element in to the Kafka.

Kafka event adapter configuration

The Kafka Event Adapter comes with a configuration file located at:

```
<DCF-Install>/Event-Processing/Kafka-Event-Adapter/<INSTANCE>/conf/kafka-event-adapter.xml
```

Sample configuration fileL

```
<kafka-event-adapter-config xmlns="http://www.watch4net.com/KafkaEventAdapter"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.watch4net.com/KafkaEventAdapter ../kafka-event-adapter.xsd ">

    <cluster>
        <!-- The Kafka bootstrap servers. -->
        <server host="127.0.0.1" port="9092" />
        <!-- uncommment the following lines, to communicate with Kafka over SSL. Kafka SSL must be
configured first.
        <additional-parameters key="security.protocol">SSL</additional-parameters>
        <additional-parameters key="ssl.keystore.location">../../../Tools/Webservice-
Gateway/Default/conf/clientkeystore</additional-parameters>
        <additional-parameters key="ssl.keystore.password">changeit</additional-parameters>
<additional-parameters key="ssl.truststore.location">../../../Tools/Webservice-
Gateway/Default/conf/truststore</additional-parameters>
        <additional-parameters key="ssl.truststore.password">changeit</additional-parameters> -->
    </cluster>

    <!-- an event reader -->

    <event-reader topic-name="events" stream="data">
        <initial-topic-partition-offset-seeker existing-reset-policy="earliest" new-reset-
policy="earliest"/>
```

```
          <topic-partition-offset-commit location="disk"/>
         <topic-partition-listener existing-reset-policy="earliest" new-reset-policy="earliest"
refresh-interval="5m"/>
    </event-reader>

    <event-reader topic-name="events" stream="data">
        <kafka-consumer-task>
            <consumer group-id="test-consumer-group" />
        </kafka-consumer-task>
    </event-reader>

    <!-- an event writer -->

    <event-writer topic-name="sam_notifications" stream="data" isJson="true" >

         <kafka-producer-settings>
         <producer
      compression-type="none"/>
         </kafka-producer-settings>
     </event-writer>
</kafka-event-adapter-config>
```

Multiple Kafka producer Configuration

User can modify event processing configuration file (processing.xml) manually to send the collected events data to multiple Kafka endpoints.

Sample example is below:

Copy existing Kafka configuration file *kafka-event-adapter.xml* to *kafka-event-adapter2.xml* in the same folder: (sample path: `<DCF-Install>/Event-Processing/Kafka-Event-Adapter/<INSTANCE>/conf/`)

Add new Kafka cluster info in *kafka-event-adapter2.xml*.

Alter configuration file *processing.xml* is like below:

```
<processing-element name="KAFKA1" enabled="true" type="Kafka-Event-Adapter" config="Kafka-Event-
Adapter/smarts-notifs-events/conf/kafka-event-adapter.xml"/>
        <processing-element name="Smarts" enabled="true" config="Smarts-Listener/smarts-notifs-
events/conf/smarts-listener.xml" data="KAFKA1 KAFKA2"/>
<processing-element name="KAFKA2" enabled="true" type="Kafka-Event-Adapter" config="Kafka-Event-
Adapter/smarts-notifs-events/conf/kafka-event-adapter2.xml"/>
```

Restart the smarts event management pack service.

Now Smarts events data will be pushed to 2 Kafka endpoints like KAFKA1 and KAFKA2.

Kafka Event Adapter parameters

Parameter	Description
<cluster>	This tag must occur at least once.
<server>	This tag must occur at least once: * **host**: The address of one of the kafka bootstrap servers * **port**: The port that the bootstrap server is listening on
<producer>	This tag is optional, but may be used for templating kafka producers. Refer to the schema for more information.
<consumer>	This tag is optional, but may be used for templating kafka consumers. Refer to the schema for more information.
<connection>	This tag is optional, but may be used for templating kafka connections. Refer to the schema for more information.
<additional-parameters>	This tag is optional, but may be used for configuring kafka options outside of the purview of the previous tags. These options include, but are not limited to SSL connection parameters.

Configuring Event Reader parameters

Parameter	Description
<event-reader>	This tag is used to define a component that will consumer from kafka: **topic-name** : This attribute is used to specify the kafka topic to read from. **stream** : This attribute is used to specify the stream to push the events to once they have been read.
<kafka-consumer-task>	This tag may be used to customize the way that we will read from kafka. * **poll-timeout-ms** : The poll timeout for kafka. * **error-timeout-ms** : The error timeout for connecting to kafka. * **error-timout-double-n** : How many errors before we double the timeout. * **error-timout-max-ms** : The maximum error timeout.
<consumer>	This tag may be used to customize how this element will consume from kafka. Of note, is that the attribute group-id must be used here if we plan on joining a kafka consumer group.
<connection>	This tag may be used to customize the kafka connection for this element.
<additional-parameters>	This tag may be used to customize the additional-parameters for this element.

Parameter	Description
<initial-topic-partition-offset-seeker>	This element is used when wanting to control how we seek in a kafka topic. **existing-reset-policy** : policy when seeking on existing partition but the requested offset no longer exists. Must be one of earliest or latest. **new-reset-policy** : policy when seeking on a new partition. Must be one of earliest or latest.
<topic-partition-offset-commit>	This element must be used if we would like to control how our offsets are committed. **location** : where to commit the topic partition offsets. Must be one of disk, kafka, none. **commit-interval**: Offsets are committed no more frequently than this interval.
<topic-partition-listener>	Required if kafka-consumer group is not set. Used to refresh the list of topic partitions periodically. **new-reset-policy** : where to seek in new partitions. (at earliest or latest offset in the new partition: usually at the earliest). **existing-reset-policy** : where to seek in existing partitions when current offset is out of range. (usually at the earliest).
refresh-interval	how often to check for new partitions.

Configuring Event Writer parameters

Parameters	Description
<event-writer>	This tag is used to define a component that will write to kafka. **topic-name** : The topic we will write to. **stream** : The event stream to consume from. **isJson** : Json format output is enabled.
<kafka-producer-settings>	This tag may be used if we want to customize how the writer writes to kafka. Refer to the schema for more information.
<connector-component-behavior>	This tag may be used if we want to control how often we flush to kafka.
<key-encoder>	This tag may be used if we want to customize how the kafka key is encoded.
<value-encoder>	This tag may be used if we want to customize how the kafka value is encoded.

Sample Kafka producer configuration (for Event Writer) values are below:

```
<kafka-producer-settings>
        <!-- Servers information -->
          <server host="10.106.126.209"/>

        <!-- Producer (and topic) information -->
        <producer topic-name="events"
            acks="1"
```

```
            retries="0"
            linger-ms="0ms"
            buffer-memory="33554432"
            compression-type="none"
            batch-size="16384"
            max-block-ms="2s"
            max-in-flight-requests-per-connection="5"
            max-request-size="1048576"/>

        <!-- All connection parameters will be attributes except "server" -->
        <connection request-timeout-ms="5s"
            connections-max-idle-ms="9m"
            retry-backoff-ms="100ms"
            reconnect-backoff-ms="50ms"/>

        <!-- Additional properties -->
        <additional-parameters key="metadata.max.age.ms">1000</additional-parameters>
        <additional-parameters key="receive.buffer.bytes">32768</additional-parameters>
        <additional-parameters key="send.buffer.bytes">131072</additional-parameters>
    </kafka-producer-settings>
```

Logging

All Event processing logs for a particular management pack will be found in below location:

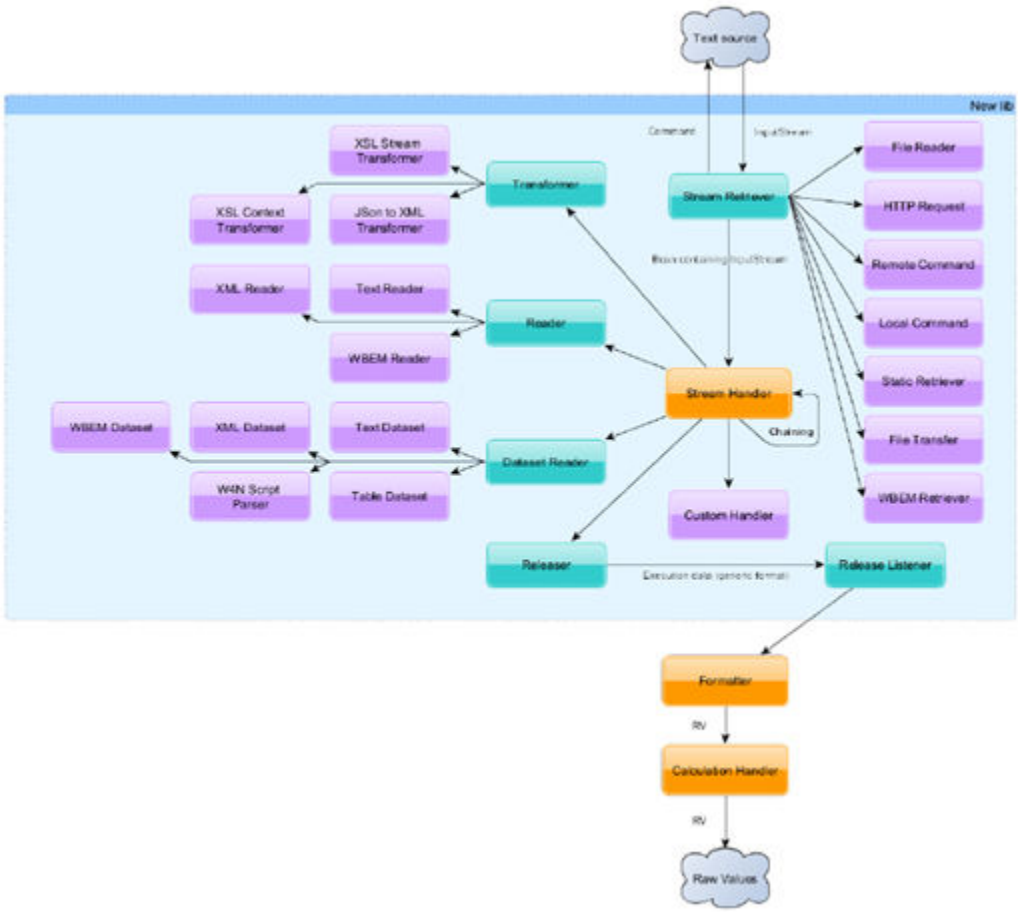
```
<DCF-Install>/Event-Processing/Event-Processing-Manager/<INSTANCE>/logs/processing-0-0.log
```

Stream Collector

5

Architecture

The Stream Collector is divided into multiple components interacting between each other in order to parse text source properly and generate raw values. Refer the figure for a global overview of the architecture.



- **Stream Retriever**: These components are used to retrieve the text sources.
- **Dataset Reader**: These components divides the text sources into smaller text chunks. This proves useful in order to be able to parse multiple small chunks in parallel.

- **Reader**: These components parses the different text source and extract information out of them and enhance the context.
- **Transformer**: These components modifies/standardize the different text sources in order to parse them.
- **Context** : This component holds the various dynamic information's required for the generation of Raw Values.
- **Releaser**: This component triggers the generation of Raw Values out of the current context.
- **Release Listener**: This component receives the generation requests from the Releaser, computes the Raw Values and send them to the next component in the collecting chain of the Collector Manager.

Configuring Stream Collector:

Following is a template of StreamCollector config and all options:

```
<collector-configuration>
    <source>Source property value</source>
    <collecting-group>Retention group</collecting-group>
    <default-character-encoding>The expected character encoding in the streams ex:UTF-8</default-
character-encoding>
    <!-- CHOICE -->
    <properties-refresh-periods>The period between a forced +r</properties-refresh-periods>
    <!-- OR -->
    <auto-detect-properties-refresh />
    <collecting-threads-pool-size>The number of threads allowed for all the    collecting
chains</collecting-threads-pool-size>
    <!-- One or more -->
    <collecting-configurations name="Configuration name">
        <!-- Zero or more -->
        <include-contexts>Location to an execution contexts file</include-contexts>
        <!-- Zero or more -->
        <execution-contexts name="Context name">
            <!-- One or more -->
            <properties name="Property name">Property value</properties>
        </execution-contexts>
        <data-retrieval-file>File containing the data retrieval chain </data-retrieval-file>
        <!-- One or more -->
        <data-listeners id="The releasing ID to listen to"
            variable-id="Optional properties for the Raw Value variable ID. Default is source device
module part name"
            variable-id-separator="Optional separator for the properties for the Raw Value variable
ID. Default is nothing"
            normalize-variable-id="Optional flag to normalize the properties for the Raw Value
variable ID. Default is true">
            <!-- Optional -->
            <timestamp
                context-key="execution context key where the value will be the timestamp"
                format="optional format when the value isn't numeric" />
            <!-- Zero or more -->
            <values context-key="execution context key where the value will be"
                type="computation type: counter (default), delta, rate or a contextualized value for
```

```
runtime selection"
                required="true (default) or false">
                <name>Optional metric name property</name>
                <unit>Optional metric unit property</unit>
                <!-- Zero or more property extraction from the value context-key value -->
                <extractions pattern="Regex containing groups">
                    <!-- One or more -->
                    <value group="regex group">property name</value>
                </extractions>
                <!-- Zero or more -->
                <replace value="value to replace" by="replacement or ommit to nullify"
                    pattern="true or false (default)" />
                <!-- Zero or more -->
                <properties context-key="execution context key where the property will be"
                    property-name="Raw Value property name">
                    <!-- Zero or more -->
                    <replace value="property to replace" by="replacement or ommit to nullify"
                        pattern="true or false (default)">
                </properties>
                <!-- Zero or more -->
                <hardcoded-properties key="property name">Property value
                </hardcoded-properties>
                <!-- Optional -->
                <dynamic-properties
                    prefix-char="The character prefixed to the dynamic properties keys in the
execution context (Default = '+')" />
            </values>
            <!-- Zero or more -->
            <dynamic-values>
                Same as value but the context-key contains a regex that will be applied
                to every execution context property name in order to extract values
            </dynamic-values>
            <!-- Zero or more -->
            <properties context-key="execution context key where the property will be"
                property-name="Raw Value property name">
                <!-- Zero or more -->
                <replace value="property to replace" by="replacement or ommit to nullify"
                    pattern="true or false (default)">
            </properties>
            <!-- Zero or more -->
            <hardcoded-properties key="property name">Property value
            </hardcoded-properties>
            <!-- Optional -->
            <dynamic-properties
                prefix-char="The character prefixed to the dynamic properties keys in the execution
context (Default = '+')" />
        </data-listeners>
    </collecting-configurations>
</collector-configuration>
```

1 Each stream collector will have a collector-configuration with following definitions:

   a *source* tag is a hardcoded value to identify the source of the data, this is added as property for each raw value.

b *collecting-group* is the retention group and will be added as a Meta property for each rawvalue.

c *properties-refresh-periods* or *auto-detect-properties-refresh* tags define when to force the action for each raw value as refreshed. *properties-refresh-periods* will force the refresh action after defined time period whereas *auto-detect-properties-refresh* will detect automatically when to force the refresh action. Once the action is detected in raw value a Meta property *action* with *value r* will be added.

d *default-character-encoding* will define character encoding used by collector.

e *collecting-threads-pool-size* will define the number threads per *collecting-configurations* defined in this collector.

f One or more *collecting-configurations* tag which defines collecting configuration of the data, explained in detail below.

Example:

```
<collector-configuration xmlns="http://www.watch4net.com/Text-Collector-Configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.watch4net.com/Text-Collector-
Configuration ../textCollectorConfiguration.xsd ">
    <source>OpenStack-Collector</source>
    <collecting-group>OpenstackGroup</collecting-group>
    <default-character-encoding>UTF-8</default-character-encoding>
    <properties-refresh-periods>10m</properties-refresh-periods>
    <collecting-threads-pool-size>20</collecting-threads-pool-size>
    <collecting-configurations name="openstack-metrics-cache">
    :
    :

</collector-configuration>
```

2 In each collecting-configuration we define following:

a *include-contexts* includes a file name which has context details of end device to collect the information.

b *data-retrieval-file* which fetches the stream, transform the stream and converts into execution context data.

c One or multiple data-listeners, which converts execution context data into raw values, explained in detail below.

Example:

```
<collecting-configurations name="openstack-metrics-cache">
    <include-contexts>conf/context-openstack.xml</include-contexts>
    <data-retrieval-file>conf/openstack-metrics-cache.xml</data-retrieval-ile>
    <data-listeners id="OPENSTACK-IMAGE" variable-id="source keystid parttype partid name">
    :
    :
</collecting-configurations>
```

3 In each data-listeners define following:

a *variable-id* tag used to build unique identification for each packet by grouping the properties. One more tag *variable-id-separator* used to separate the properties. This value is added as meta property for each rawvalue.

Example:

```
<data-listeners id="SMARTS-TOPO-FASTDXA" variable-id="type Name" variable-id-separator="::">
```

Where type is *VirtualMachine* and Name as *VM-1* name for this rawvalue is updates as VirtualMachine::VM-1.

b One or more *values* tag which is a metric for rawvalue will be updated from context. This should be always a float value. If it is not a float then use replace tag to change accordingly. For each values we should provide *unit* value as shown in below example which is the unit by which this metric is measured.

Example:

Here user can map "disabled" key in context as "Availability" metric in outgoing rawvalue

```
values context-key="disabled">
                    <name>Availability</name>
                        <unit>%</unit>
     </values>
     Output:
 "metrics" : {
    "Availability" : {
      "properties" : {
        "unit" : "%",
        "name" : "Availability"
      },
      "value" : 100.0
    }
  }
```

This example is same as first one but in this case value of disabled is Boolean, to make it as metric we are converting as float using *replace* tag shown below:

```
<values context-key="disabled">
                    <name>Availability</name>
                    <unit>%</unit>
               <replace value="false" by="100"/>
               <replace value="true" by="0"/>
            </values>
```

This example is same as second one but in this case value of enabled is String, to make it as metric we are converting as float using *replace* tag shown below. If value is True then replace with 100 else if it matches regex pattern replace with 0.

```
<values context-key="enabled">
                    <name>Availability</name>
                    <unit>%</unit>
                    <replace value="True" by="100"/>
                    <replace value=".+"   by="0" pattern="true"/>
          </values>
```

c Zero or more *dynamic-values* which will get the metrics value for rawvalue from context upon regex match. To extract metric values dynamically we have to update execution context by appending a specific pattern (ex: ~@M) for each context key which should be treated as metric during the data retrieval and transformer time. And extract the values using extraction pattern as shown in example below.

Example: In execution context two values are updated with dynamic value prefix i.e ~@MAvailability=100, ~@MUtilization=100, then for the following configuration.

```
<dynamic-values context-key="~@M.*" required="false">
                <unit>%</unit>
                <extractions pattern="~@M(.+)">
                    <value group="1">name</value>
                </extraction>
</dynamic-values>
Output:
"metrics" : {
    "Availability" : {
      "properties" : {
    "unit" : "%",
       "name" : "Availability"
     },
   "value" : 100.0
},
                  "Utilization" : {
  "properties" : {
     "unit" : "%",
       "name" : "Utilization"
     },
    "value" : 100.0
   }
}
```

d Zero more *properties* which is a property for rawvalue, will be updated from context based on context-key provided.

Example:

Maps "CreationClassName" key to "type" in raw value

```
<properties context-key="CreationClassName" property-name="type"/>
```

Maps "pwrstate" key to PWRState by replacing actual integer value to string.

```
<properties context-key="pwrstate" property-name="PWRState">
                <replace value="0" by="Unknown"/>
                <replace value="1" by="Running"/>
                <replace value="3" by="Paused"/>
                <replace value="4" by="Shutdown"/>
                <replace value="6" by="Crashed"/>
                <replace value="7" by="Suspended"/>
                <replace value=".+" by="Unknown" pattern="true"/>
 </properties>
```

e Zero more *hardcoded-properties* which is a property for rawvalue will be updates directly the hardcoded value.

Example:

```
<hardcoded-properties key="devtype">CloudService</hardcoded-properties>
```

f Optional Zero or more *dynamic-properties* will consider and add context-key as properties if it is prefixed with character configured. Default prefix-char is "+".

Example:

In this example if any context key is prefixed with "@" is considered as property and added as part of property of rawvalue.

```
<dynamic-properties prefix-char=@/>
```

Example: Data-listener configuration example.

```
<data-listeners id="OPENSTACK-IMAGE" variable-id="source keystid parttype partid name">
            <values context-key="Status">
                <name>Status</name>
                <unit>code</unit>
                <replace value="active" by="0"/>
                <replace value="queued" by="1"/>
                <replace value="saving" by="2"/>
                <replace value="deleted" by="3"/>
                <replace value="pending_delete" by="4"/>
                <replace value="killed" by="5"/>
            </values>
            <dynamic-values context-key="~@M.*" required="false">
                <unit>bytes</unit>
                <extractions pattern="~@M(.+)">
                    <value group="1">name</value>
                </extractions>

            </dynamic-values>

            <properties context-key="cformat" property-name="cformat"/>

                <properties context-key="isprotec" property-name="isprotec">
```

```
                <replace value="false" by="No"/>
                <replace value="true" by="Yes"/>
            </properties>
            <properties context-key="ispublic" property-name="ispublic">
                <replace value="private" by="No"/>
                <replace value="public" by="Yes"/>
            </properties>
            <properties context-key="updated" property-name="updated"/>
            <hardcoded-properties key="datagrp">OPENSTACK-IMAGE</hardcoded-properties>
            <hardcoded-properties key="devtype">CloudService</hardcoded-properties>
            <hardcoded-properties key="parttype">Image</hardcoded-properties>
        </data-listeners>
```

Example of a Stream Collector configuration

In this example a collecting configuration is configured with 2 data-listeners. These collecting configuration will be executed against the device context provided in file *conf/context-openstack.xml* which includes all device credentials and context variables to collect the information from device.

All the retrieval information of the data is defined in file *conf/openstack-metrics-main.xml* which will get the stream, transform the stream and provide it to datalistner.

DataListner converts the contexts data available to raw values and publish it to next component defined in *collecting.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<collector-configuration xmlns="http://www.watch4net.com/Text-Collector-Configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.watch4net.com/Text-Collector-
Configuration ../textCollectorConfiguration.xsd ">
    <source>OpenStack-Collector</source>
    <collecting-group>group</collecting-group>
    <default-character-encoding>UTF-8</default-character-encoding>
    <properties-refresh-periods>10m</properties-refresh-periods>
    <collecting-threads-pool-size>20</collecting-threads-pool-size>
    <collecting-configurations name="openstack-metrics-main">
        <!--File location which has context details of end device -->
        <include-contexts>conf/context-openstack.xml</include-contexts>

     <!--File which retrieves and transforms the stream in to context data-->
        <data-retrieval-file>conf/openstack-metrics-main.xml</data-retrieval-file>

        <data-listeners id="GET-OPENSTACK-HYPERVISORS" variable-id="type device">
            <values context-key="CurrentWorkload">
                <name>CurrentWorkload</name>
                <unit>nb</unit>
            </values>
            <values context-key="RunningVMs">
                <name>RunningVMs</name>
                <unit>nb</unit>
            </values>
            <values context-key="TotalVCpus">
                <name>TotalVCpus</name>
```

```
            <unit>nb</unit>
            <properties context-key="cpu-partmod" property-name="partmod"/>
            <properties context-key="cpu-partvndr" property-name="partvndr"/>
            <hardcoded-properties key="part">System</hardcoded-properties>
            <hardcoded-properties key="parttype">Processor</hardcoded-properties>
        </values>
        <values context-key="UsedVCpus">
            <name>UsedVCpus</name>
            <unit>nb</unit>
            <properties context-key="cpu-partmod" property-name="partmod"/>
            <properties context-key="cpu-partvndr" property-name="partvndr"/>
            <hardcoded-properties key="part">System</hardcoded-properties>
            <hardcoded-properties key="parttype">Processor</hardcoded-properties>
        </values>
        <values context-key="CurrentUtilization">
            <name>CurrentUtilization</name>
            <unit>%</unit>
            <hardcoded-properties key="part">Physical Memory</hardcoded-properties>
            <hardcoded-properties key="parttype">Memory</hardcoded-properties>
        </values>
        <values context-key="FreeMemory">
            <name>FreeMemory</name>
            <unit>MB</unit>
            <hardcoded-properties key="part">Physical Memory</hardcoded-properties>
            <hardcoded-properties key="parttype">Memory</hardcoded-properties>
        </values>
        <values context-key="TotalMemory">
            <name>TotalMemory</name>
            <unit>MB</unit>
            <hardcoded-properties key="part">Physical Memory</hardcoded-properties>
            <hardcoded-properties key="parttype">Memory</hardcoded-properties>
        </values>
        <values context-key="UsedMemory">
            <name>UsedMemory</name>
            <unit>MB</unit>
            <hardcoded-properties key="part">Physical Memory</hardcoded-properties>
            <hardcoded-properties key="parttype">Memory</hardcoded-properties>
        </values>
        <values context-key="TotalDisk">
            <name>TotalDisk</name>
            <unit>GB</unit>
            <hardcoded-properties key="part">Physical Memory</hardcoded-properties>
            <hardcoded-properties key="parttype">Disk</hardcoded-properties>
        </values>
        <values context-key="UsedDisk">
            <name>UsedDisk</name>
            <unit>GB</unit>
            <hardcoded-properties key="part">System</hardcoded-properties>
            <hardcoded-properties key="parttype">Disk</hardcoded-properties>
        </values>
        <properties context-key="fqdn" property-name="device"/>
        <properties context-key="fqdn" property-name="fqdn"/>
        <properties context-key="ip" property-name="ip"/>
        <properties context-key="host" property-name="host"/>
        <properties context-key="KEYSTONE_ID" property-name="keystid"/>
```

```
            <properties context-key="TotalVCpus" property-name="nbcpu"/>
            <properties context-key="OPST_HOST" property-name="osendpt"/>
            <hardcoded-properties key="datagrp">OPENSTACK-HYPERVISOR</hardcoded-properties>
            <hardcoded-properties key="devtype">Hypervisor</hardcoded-properties>
            <hardcoded-properties key="type">HypervisorMonitor</hardcoded-properties>
        </data-listeners>

        <data-listeners id="OPENSTACK-VM" variable-id="type devid">
            <values context-key="RootCapacity">
                <name>Capacity</name>
                <unit>GB</unit>
                <properties context-key="root-natvolnm" property-name="natvolnm"/>
                <hardcoded-properties key="part">Root</hardcoded-properties>
                <hardcoded-properties key="partdesc">Root Disk for @{device}</hardcoded-properties>
                <hardcoded-properties key="parttype">Virtual Disk</hardcoded-properties>
                <hardcoded-properties key="voltype">Root</hardcoded-properties>
            </values>
            <properties context-key="avzone" property-name="avzone"/>
            <properties context-key="created" property-name="created"/>
            <properties context-key="device" property-name="device"/>
            <properties context-key="deviceid" property-name="devid"/>
            <properties context-key="deviceid" property-name="deviceid"/>
            <properties context-key="flavname" property-name="flavname"/>
            <properties context-key="flavorid" property-name="flavorid"/>
            <properties context-key="hypervsr" property-name="hypervsr"/>
            <properties context-key="imageid" property-name="imageid"/>
            <properties context-key="imagenm" property-name="imagenm"/>
            <properties context-key="KEYSTONE_ID" property-name="keystid"/>
            <properties context-key="ip" property-name="ip"/>
            <properties context-key="keyname" property-name="keyname">
                <replace value="null" by="N/A"/>
            </properties>
            <properties context-key="TotalVCpus" property-name="nbcpu"/>
            <properties context-key="OPST_HOST" property-name="osendpt"/>
            <properties context-key="projid" property-name="projid"/>
            <properties context-key="pwrstate" property-name="pwrstate">
                <replace value="0" by="Unknown"/>
                <replace value="1" by="Running"/>
                <replace value="3" by="Paused"/>
                <replace value="4" by="Shutdown"/>
                <replace value="6" by="Crashed"/>
                <replace value="7" by="Suspended"/>
                <replace value=".+" by="Unknown" pattern="true"/>
            </properties>

            <properties context-key="status" property-name="status"/>
            <properties context-key="updated" property-name="updated"/>
            <properties context-key="userid" property-name="userid"/>
            <hardcoded-properties key="devtype">VirtualMachine</hardcoded-properties>
            <hardcoded-properties key="datagrp">OPENSTACK-VM</hardcoded-properties>
            <hardcoded-properties key="type">VirtualMachine</hardcoded-properties>
        </data-listeners>
    </collecting-configurations>
</collector-configuration>
```

Smarts Collector

6

This chapter includes the following topics:

- Overview
- Configuration
- Logging

Overview

Smarts Collector is a high performance, asynchronous process, which gather data from Smarts in-memory repository very efficiently. It retrieves data from any Smarts domain and push the same to Kafka in a particular JSON format.

The installation and configuration of this management pack happens through DCF controller. This section gives an overview of how each configuration files configured and used. So that user can alter the same as per the need.

Configuration

After installing the management pack which use this collector, all configuration files related to this collector is available at :`<DCF-Install>/ Collecting/Smarts-Collector/<Management-Pack>/conf/`

To use this collector, simply declare it in the collecting configuration file: `<DCF-Install>/ Collecting/Collector-Manager/<Management-Pack>(smarts-metrics-collect)/conf/collecting.xml`

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="http://www.watch4net.com/APG/Collecting" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.watch4net.com/APG/Collecting collecting.xsd ">
<connectors>
<connector enabled="true" name="File" type="File-Connector"   config="conf/file-connector.xml" />
<connector enabled="true" name="Kafka" type="Kafka-Connector"   config="Kafka-Connector/smarts-metrics-collect/conf/kafka-connector.xml" />
```

```
</connectors>

<collectors>
<collector enabled="true" name="smarts-INCHARGE-AM-PM-0-metrics" next="Kafka File" type="Smarts-
Collector" config="Smarts-Collector/smarts-metrics-collect/conf/smarts-INCHARGE-AM-PM-0-metrics.xml" />
<collector enabled="true" name="smarts-INCHARGE-AM-PM-0-topo" next="Kafka File" type="Smarts-
Collector" config="Smarts-Collector/smarts-metrics-collect/conf/smarts-INCHARGE-AM-PM-0-topo.xml" />
</collectors>
</config>
```

There are 2 components here. Collectors section deals with configuration of domain managers and the data which need to be pulled from these servers.

Connectors section deals with sending this collected data (JSON format) to Configured Kafka servers and/or to a file for debugging.

Configuration files Information

These configuration files deals with polling Smarts topology.

1 smarts-INCHARGE-AM-PM-0-metrics.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "smarts.dtd">
<config>
<domain>INCHARGE-AM-PM</domain>
<host>localhost:12345</host>
<username>admin</username>
<password>{38985215010FA2D2BCBDCCE151D4C261A68B1E0486B804084A615512D4BAB69C56EEFAA581451988199D8314
585605ED}</password>
<group>group</group>
<period>240</period>
<thread pool-size="3" />
<dm-connection pool-size="3" keep-connection="false" creation-grace-time="100" />
<indicators select="both">conf/pm-metrics.xml</indicators>
<properties refresh="00:00/86400" send-on-refresh-only="false" />
<smooth-factor>0.5</smooth-factor>
</config>
```

The polling configuration file (typical path : `<DCF Install>/Collecting/Smarts-Collector/<Management-Pack>(smarts-metrics-collect)/conf/smarts-INCHARGE-AM-PM-0-metrics.xml`) will have details regarding Smarts domain manager connection parameters and also points to the metric configuration file which have configurations regarding what details need to be fetched from configured Smarts domain manager. Here that file is *pm-metrics.xml*.

The **domain** attribute let you specify Smarts Domain manager name.

The **host** attribute is host name and port information of the Smarts server.

The **username** and **password** tag are User Name and password used to login to Smarts server.

The **group** tag value used by the collector to assign the string specified in the group element to generated metric.

The **period** is the polling period.

The **thread pool-size** attribute let you specify the number of concurrent polling threads which will share a customizable number of connections to Smarts InCharge domain (specified in dm-connection pool-size).

Since the Collector is design to be very fast, it can disturb Smarts InCharge domain: that's why there are two other parameters which control the polling rate.

The **dm-connection creation-grace-time** is the minimum delay between consecutive connections to the Smarts InCharge domain.

The **smooth-factor** control the actual polling rate.

For example, if polling period is 240 second and smooth factor is 0.5, the Collector will try to pool the domain in 0.5 * 240 = 120 seconds.

The **dm-connection keep-connection** attribute control whether we should keep domain connections after the polling process or not. It is usually more reliable to set it to false ensuring that we always have proper connectivity to the domain manager.

The d**m-connection** no-clobber attribute controls the connection creation process: if set to true, no new connection are created until the next polling cycle starts. This behavior may be useful in certain cases to avoid connection pile-ups on domain managers. Should be left to false, unless this problem actually occurs, as enabling it will lead to some domain managers not being polled when random connection timeout occurs. This attribute is optional (default value is false).

Finally, the most important configuration element of a Smarts InCharge Collector is its domain descriptor file which defines what is interesting in a Smarts InCharge domain. You specify it in the **indicators** element and the syntax of this file is described in the following section.

```
<indicators select="both">conf/pm-metrics.xml</indicators>
```

The select attribute enables you to choose between 3 polling modes:

properties polla only properties, not generating any actual performance metrics. This is useful for inventory purpose.

values generatea only the performance values and not poll properties at all.

both is the default polling mode, polling both properties and values.

a pm-metrics.xml (Domain descriptor file)

This domain descriptor is an XML file which describes what will be retrieved from a specific Smarts InCharge domain.

Example 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE domain SYSTEM  "indicators.dtd">
<domain>
<script id="datagrp-tag">
<![CDATA[
```

```
ctx.create("datagrp", "SMARTS-PM-METRICS");
]]>
</script>

<data id="Device" class="UnitaryComputerSystem" start="true">
<scripted-property script-id="datagrp-tag" />
<property name="IPAddress Description Model Location SNMPAddress PrimaryOwnerContact Vendor"
rename="ip devdesc model location ip contact vendor" />
<property name="CreationClassName" rename="type" />
<property name="IsManaged" rename="ismanaged" />
<value name="IsUnresponsive" rename="Reachability" unit="%">
<transformation match="false" replace="100.0" />
<transformation match="true" replace="0.0" />
</value>
</data>
```

The **data** element is equivalent to a class of Smarts Domain manager. You have to create data block for each interested class of Smarts domain manager.

The nested **value** and **property** elements specify which numerical attributes and which object properties should be retrieved.

Both **value** and **property** elements can have a **rename** attribute to rename the Smarts InCharge attribute. The **value** supports a special attribute called **unit** which enables you to specify the value unit.

Note that the **start** attribute. It is VERY important not to forget it. It is used by the collector as an “entry point” into Smarts InCharge object repository. When the collector starts the polling process, it will retrieve all the instances of the classes whose start attribute is set to true.

The above example:

1. fetch instances of UnitaryComputerSystem.

2. Each element properties as per property declaration. For ex: CreationClassName attribute will be renamed as type .. etc

3. Datagrp is added as new property on the output as per script output.

4. IsUnresponsive metric is renamed as Reachability and its values are mapped to 100.0% or 0% if it value is true or false respectively.

Finally JSON object will be constructed and pushed to configured Kafka.

Example 2:

```
<data class="NetworkAdapter" containment="InstrumentedBy" start="true">
<scripted-property script-id="datagrp-tag" />
<property name="CreationClassName" rename="type" />
<property name="IsManaged" rename="ismanaged" />
</data>

<data class="NetworkAdapter_Performance" instance-name-pattern="@PREVIOUS">
<value name="ifInDiscardsRate ifOutDiscardsRate"
rename="ifInDiscards ifOutDiscards" unit="Pkts/s" />
```

```
<value name="ifInErrorsRate ifOutErrorsRate"
rename="ifInErrors ifOutErrors" unit="Pkts/s" />
<value name="ifInNUcastPktsRate ifOutNUcastPktsRate ifInUcastPktsRate ifOutUcastPktsRate"
rename="ifInNUcastPkts ifOutNUcastPkts ifInUcastPkts ifOutUcastPkts"
unit="Pkts/s" />
<value name="CurrentUtilization ifInOctetsRate ifOutOctetsRate MaxSpeed" unit="float" />
</data>
```

The **instance-name-pattern** attribute allows you to construct a new instance identifier instead of default value. This helpa in case the object name is getting changed frequently in Smarts domain manager.

Smarts InCharge stores the topology using many objects linked together through relations. The collector can use these relations to explore the topology. It enables you to collect properties and values from many linked objects from the instances of a given class. You can do this by specifying a space separated list of relation fields in the containment attribute of the data element. Relation fields can be either single reference or reference sets.

In this above example:

1. The collector explores the **InstrumentedBy** relation of the NetworkAdapter instances (i.e. declared in a data block without start attribute or with start="false").

2. If the InstrumentedBy contains some NetworkAdapter Performance instances, the collector will retrieve the ifInDiscardsRate and other attributes declared under **value** from the Smarts InCharge object.

3. Only related NetworkAdapter Performance instances of Interface class will be retrieved and not all instances will be retrieved.

4. The generated value will keep any previously assigned properties like **type**.

2 smarts-INCHARGE-AM-PM-0-topo.xml

This topology configuration file and its descriptor file below are similar to other configuration files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "smarts.dtd">
<config>
<domain>INCHARGE-AM-PM</domain>
<host>localhost:12345</host>
<username>admin</username>
<password>{38985215010FA2D2BCBDCCE151D4C261A68B1E0486B804084A615512D4BAB69C56EEFAA581451988199D8314
585605ED}</password>
<group>group</group>
<period>3600</period>
<thread pool-size="3" />
<dm-connection pool-size="3" keep-connection="false" creation-grace-time="100" />
```

```
<indicators select="both">conf/pm-topo.xml</indicators>
<properties refresh="00:00/86400" send-on-refresh-only="false" />
<smooth-factor>0.5</smooth-factor>
</config>
```

a pm-topo.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE domain SYSTEM  "indicators.dtd">
<domain>
<script id="datagrp-tag">
<![CDATA[
ctx.create("datagrp", "SMARTS-PM-TOPO");
]]>
</script>
<data id="Device" class="UnitaryComputerSystem" containment="ComposedOf PartOf" start="true">
<scripted-property script-id="datagrp-tag" />
<property name="IPAddress Description Model Location SNMPAddress PrimaryOwnerContact Vendor"
rename="ip devdesc model location ip contact vendor" />
<property name="CreationClassName" rename="type" />
<property name="IsManaged" rename="ismanaged" />
<value name="IsUnresponsive" rename="Reachability" unit="%">
<transformation match="false" replace="100.0" />
<transformation match="true" replace="0.0" />
</value>
</data>

<data class="Memory" containment="PartOf" start="true">
<scripted-property script-id="datagrp-tag" />
<property name="CreationClassName" rename="type" />
<property name="IsManaged" rename="ismanaged" />
<value name="DeviceID" rename="id" unit="integer" />
</data>

<data class="FileSystem" containment="PartOf" start="true">
<scripted-property script-id="datagrp-tag" />
<property name="CreationClassName" rename="type" />
<property name="IsManaged" rename="ismanaged" />
<value name="DeviceID" rename="id" unit="integer" />
</data>
```

These configuration files will retrieve Smarts domain manager topology information based on the configured interval.

Configuration file properties

Below are the list of XML tags which can be used in configuration files for polling attributes from Smarts domain manager.

Scope

By default, polled properties are visible within their own data block and every other block we may reach from it. A data block can override a property and then shadow any previous value. When the collector exits a data block, it will normally discard the collected properties.

It is sometimes useful to retrieve properties from other instances using relationships, and tag local numerical attributes with them. That is why you can set the keep-properties attribute to true in order to extends the property lifetime to the previously explored data block.

```
<data class=”Interface” containment=”Underlying InstrumentedBy” start=”true”>
….
…..

<data class=”NetworkAdapter Performance”>
<value name=”ifInOctetsRate” rename=” ifInOctets ” unit=”Octets/s”/>
</data>

<data class="IP" keep-properties="true">
<scripted-property script-id="datagrp-tag" />
<property name="Address" rename="ifaceip" append-with="," />
<value name="RoundTripTime" rename="ResponseTime" unit="ms" />
</data>
```

In this example, we are exploring the Underlying relationship to get the interface IP address and tag ifInOctets values with.

Qualified relations: If the relation attributes contains the string no-values this will give a hint to the collector not to explore this relation when property polling is disabled. This is useful when some classes are explored only to gather properties.

```
<domain>
<!-- ... -->
<data class=”Interface” containment="Underlying(ip-no-values)">
<!-- ... -->
</data>
<!-- ... -->
</domain>
```

Restricted relation sets: A relation can be either a single reference, or a reference set. If we want to process a reference set as a single reference, we can restrict the collector to retrieve only the first instance. This is achieved by adding a [0] at the end of the relation attribute name.

```
<domain>
<!-- ... -->
<data class=”Host” containment="ComposedOf[0]" start=”true”>
<!-- ... -->
</data>
<!-- ... -->
</domain>
```

In the example, we would only retrieve the first component from each Host instance. Note that the [0] suffix is not part of a qualified relation name.

Special relations: Special relation which doesn't exist in Smarts manager. This relation is called self and enables you to re-explore a class instance within another context.

```
<domain>
<!-- ... -->
<data class="Interface" containment="self(to-another-context)">
<!-- ... -->
</data>
<!-- ... -->
</domain>
```

Conditional Relationship Exploration: Sometimes you don't want to explore all the relations anytime, or don't want to get all the instances from a relation. That why the from attribute has been introduced to the data element. With this attribute, you can choose in what context you want to explore a class instance.

```
<domain>
<!-- ... -->
<data id="Device" class="Host" containment="ComposedOf" start="true">
<!-- ... -->
</data>
<data id="Device" class="Router" containment="ComposedOf" start="true">
<!-- ... -->
</data>
<data class="Processor" containment="InstrumentedBy" from="Host">
<!-- ... -->
</data>
<data class="Interface" containment="InstrumentedBy" from="Device">
<!-- ... -->
</data>
<!-- ... -->
</domain>
```

The from attribute can contain a space separated list of data block class names or data block ids. In the previous example, we would explore Processor block from Hosts only while we would explore Interfaces for every Device. Please note that the class names specified in the from attribute WILL NOT HONOR CLASS HIERARCHY. It must be the exact name specified in the class attribute of the data block the collector comes from.

When several choices are involved, the following rules are applied:

1. The collector firsts locates matching data block (with or without from attribute).
2. If blocks with matching id are found, we explore them.
3. If there are no block with matching id but some with matching class names, we explore them.
4. If no matching blocks were found in 2 and 3, we explore default matches (without from attribute).

Contextual Relationship Exploration: The from attribute can contain even more specific references. The id or class name specified can be appended a qualified relation name (within brackets). The data block will be evaluated according the rules described in the previous section, but only if the collector is coming from the specified relation. This is particularly useful with the self relation to avoid infinite recursion.

```
<domain>
<!-- ... -->
<data class="Interface" containment="self(special-context)">
<!-- ... -->
</data>
<data class="Interface" from="Host[self(special-context)]">
<!-- ... -->
</data>
<!-- ... -->
</domain>
```

Property polling and processing

1. Basic Information: A transformation element nested into a property one performs a simple search and replace in the collected attribute. You have to specify two attributes match and replace which are respectively a regular expression and the new content to replace the match with. They are always all orderly evaluated and then a value can be transformed several times.

The property element also support a drop-empty attribute which default value is true, meaning that empty properties will be discarded.

```
domain>
<!-- ... -->
<data class="Interface">
<property name="MaxSpeed" rename="maxspeed" drop-empty="true">
<transformation match="^([0-9]{2,})[0-9]{6}$" replace="$1 Mbits"/>
<transformation match="^([0-9]{2,})[0-9]{3}$" replace="$1 Kbits"/>
</property>
</data>
<!-- ... -->
</domain>
```

Here, we collect the MaxSpeed attribute from Interface instances transforming it into a more human readable string. Any empty speed value will be discarded. If we specified the force tag in property declaration, any other value will result into an uninitialized value.

2. Multiple/Tabular properties handling: The default behavior of property polling is to always keep the first value for each distinct property name within a data block. However, if multiple values should be concatenated, you can specify the append- with attribute. Instead of being discarded, additional values will be appended to the previous one using the content of the append-with as a separator.

Note that this also applies with the keep-properties set to true, when already existing properties are retrieved from relation.

```
<domain>
<!-- ... -->
<data class="HostSystem">
<property name="Vendor OperatingSystem OSLevel"
rename="devdesc devdesc devdesc" append-with=" " />
</data>
<!-- ... -->
</domain>
```

In this example, we will append Vendor, OperatingSystem and OSLevel with a <space> character to generate a nice devdesc property.

Value polling and processing

1. Simple sanity checks: The value elements support two attributes min and max which provide simple sanity check. If a value is outside specified boundaries, it will be discarded (unless the force attribute has been specified).

```
<domain>
<!-- ... -->
<data class="NetworkAdapter Performance">
<value name="ifInOctetsRate" unit="Octets/s" min="0"/>
</data>
<!-- ... -->
</domain>
```

This example would drop negative (and obviously invalid) rates. Since we don't specify max attribute, any positive value will be accepted.

2. Inlined expressions: When specifying the source attribute for a value using the name attribute (e.g. name = "ifInOctet- sRate"), you can also write some simple expressions. Most arithmetic operators are supported (+, -, *, /). Note that you can not use <space> in an expression since it would be splitted in two values: you must quote it first.

```
<domain>
<!-- ... -->
<data class="HostPhysicalDevice">
<value name="'NoOfBlocks*BlockSize'" rename="Capacity" unit="Bytes"/>
</data>
<!-- ... -->
</domain>
```

Since the capacity in bytes in not available here, we multiply the number of available block by the block size to compute it.

Embedded scripts:Sometimes the previously described features are not flexible enough to poll a Smarts domain. Some models can be very complex, requiring some advanced processing. This processing can be achieved embedding processing scripts in a domain descriptor.

1. Script definition: Scripts are declared in separate script elements. For each script you have to declare its id, its parameters (in) and its output (out). The script body in declared in the script elements itself.

```
<domain>
<!-- ... -->
<script id="my-script" in="Param1 Param2" out="Output1">
<![CDATA[
<!-- script body -->
]]>
</script>
<!-- ... -->
</domain>
```

Scripting engine is based on Apache JEXL. The syntax is mostly Java based with some handy shortcuts but some restrictions. For more information on the exact syntax, please refer to http://jakarta.apache.org/commons/jexl/reference/syntax.html.

2. Script reference: A script can be referenced in a data block:

- to process values using a scripted-value element

- to process properties using a scripted-property element

- to filter instances using pre-filter-script element and/or post-filter-script element

Each of these elements must specify the script reference using the script-id attribute. They must also specify input using the in attribute (except for pre-filter-script) and output using the out attribute (for scripted-value and scripted-property only).

Input is a list of space separated Smarts InCharge attribute names while output is a list of DCF names. They are similar to the name and rename attributes in value and property elements. If they are not or partially specified, those declared in script definition will be used.

```
<domain>
<!-- ... -->
<data class="... ">
<!-- ... -->
<scripted-value script-id="my-script" in="MyInAttribute" out="MyVar"/>
<!-- ... -->
</data>
<!-- ... -->
</domain>
```

3. Script variables: Each script is evaluated in its own context. As in any programming language, variables can be referenced using identifiers. When a script is evaluated, each input parameter is defined as a read-only variable (in can be shadowed by a local variable however).

Input parameters are either Smarts InCharge repository objects directly gathered from domain using the Remote API (see reference for details) or constants (when enclosed by quotes in the input parameter list). A script will be exposed all attributes collected from the domain, not only those requested as input parameters.

A script can obviously use its own variables. Furthermore, it can access some special variables:

- ctx is the script context. Most functionality is based on this special variable. Depending on its purpose, a script ctx variable will expose different functionality.

- smInstance is the instance name from Smarts InCharge.

- apgInstance is the instance name from DCF (it may be equal to smInstance)

- pctx is the polling context. It enables a script to interract with poller internals. You should never use it unless you know what you are doing.

4. General script context: The script context offer some helper methods for scripts. In any script you can access the following methods from the ctx variable:

Methods	Description
Object value(value)	Unpacks a Remote API object.
Object[] array(value)	Unpacks a Remote API array.
String str(value)	Creates a string from the specified object, the returned string is never null but can be empty.
String[] strarray(value)	Creates a string array from the specified object, the returned string is never null but can contain empty strings.
String nstr(value)	Creates a string from the specified object, the returned string can be null.
String[] nstrarray(value)	Creates a string array from the specified object, the returned string can't be null but can contain null values.
Number num(value)	Creates a number from the specified object, the returned number can't be null but can be NaN .
Number[] numarray(value)	Creates a number array from the specified object, the returned number can't be null but can contain some NaN .
Number nnum(value)	Creates a number from the specified object, the returned number can be null.
Number[] nnumarray(value)	Creates a number array from the specified object, the returned number can't be null but can contain null values.
void id(instance, key, suffix)	Creates a unique id from the specified names (instance and suffix can be null).
void store(pctx, id, value)	Stores a value in the polling context using the specified unique id.
Object fetch(pctx, id)	Retrieve a value from the polling context using the specified unique id.
Object timestored(pctx, id)	Retrieves the timestamp when a value was stored from the polling context using the specified unique id.
Object looptime- stored(pctx, id)	Retrieves the polling cycle timestamp when a value was stored from the polling context using the specified unique id.
void store(key, value)	Stores a value in the current polling context using the specified key. Keys will use input parameter mappings.
Object fetch(key)	Retrieves a value from the current polling context using the specified key.

Object timestored(key)	Retrieves the timestamp when a value was stored from the current polling context using the specified key.
Object looptime- stored(key)	Retrieves the polling cycle timestamp when a value was stored from the current polling context using the specified key.
void pstore(key, value)	Stores a value in the previous polling context using the specified key. Keys will use input parameter mappings.
Object pfetch(key)	Retrieves a value from the previous polling context using the specified key.
Object ptimestored(key)	Retrieves the timestamp when a value was stored from the previous polling context using the specified key.
Object plooptime- stored(key)	Retrieves the polling cycle timestamp when a value was stored from the previous polling context using the specified key.
void istore(key, value)	Stores a value in the initial polling context (which is kept accross polling loops) using the specified key. Keys will use input parameter mappings.
Object ifetch(key)	Retrieves a value from the initial polling context (which is kept accross polling loops) using the specified key.
Object itimestored(key)	Retrieves the timestamp when a value was stored from the initial polling context (which is kept accross polling loops) using the specified key.
Object ilooptime- stored(key)	Retrieves the polling cycle timestamp when a value was stored from the initial polling context (which is kept accross polling loops) using the specified key.
void out(value)	Prints the value on standard output.
void err(value)	Prints the value on error output
void log(value)	Prints the value in logs
int time()	Returns the current timestamp in seconds.
int looptime()	Returns the current polling cycle timestamp in seconds

Scripted values are scripts which generate values. Values are generated with the create methods. Every created value will be automatically sent without any other operation unless discarded with the discard method. These scripts can access the following methods from ctx:

Methods	Description
Value create(name)	Creates an uninitialized value using the specified name. It is equivalent to create(name, NaN) .
Value create(name, value)	Creates a value using the specified name. It is equivalent to create(name, value, null) .

Value create(name, value, unit)	Creates a value using the specified name and unit. If script reference defines output mapping, the name will be transformed to the reference output name.
void discard(value)	Discards a previously created value

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE domain SYSTEM ”indicators.dtd”>
<domain>
<!-- ... -->
<script id=”capacity-decode” in=”Capacity” out=”Capacity”>
<![CDATA[
v = ctx.str(Capacity);
if(v.matches(”([0-9n.]+)( GB)?”)) {
ctx.create(”Capacity”,
v.replaceAll(”([0-9n.]+)( GB)?”, ”$1”)*1024*1024*1024,
”Bytes”);
} else if(v.matches(”([0-9n.]+)( MB)?”)) {
ctx.create(”Capacity”,
v.replaceAll(”([0-9n.]+)( MB)?”, ”$1”)*1024*1024,
”Bytes”);
} else if(v.matches(”([0-9n.]+)( KB)?”)) {
ctx.create(”Capacity”,
v.replaceAll(”([0-9n.]+)( KB)?”, ”$1”)*1024,
”Bytes”);
} else if(v.matches(”([0-9n.]+) *”)) {
ctx.create(”Capacity”,
v.replaceAll(”([0-9n.]+) *”, ”$1”)*1,
”Bytes”);
}
]]>
</script>
<!-- ... -->
</domain>
```

This sample script convert a capacity string into a numeric value. The capacity string is expected to be a number followed by a unit. Please note the implicit string to number conversion in v.replaceAll("([0-9\.]+) *", "$1")*1.

The script may be applied with the following reference:

```
<scripted-value script-id=”capacity-decode” in=”TotalSize” out=”TotalSize” />
```

The Capacity parameter will be dynamically mapped to TotalSize attribute. The created value will be automatically renamed from "Capacity" to TotalSize. Note that the mappings doesn't prevent script from creating more value, however, they will not be automatically renamed.

Another usefull example is the delta script. It shows how you can store data across polling cycles to compute deltas.

```
<domain>
<!-- ... -->
<script id="rate" in="value unit " out="rate">
<![CDATA[
previousValue = ctx.ifetch(rate);
previousTimestamp = ctx.itimestored(rate);
currentValue = ctx.num(value);
currentTimestamp = ctx.time();
if (previousValue != null) {
newRate = (currentValue-previousValue) / (currentTimestamp-previousTimestamp);
if (newRate >= 0) {
ctx.create(rate, newRate, unit);
}
}
ctx.istore(rate, currentValue);
]]>
</script>
<!-- ... -->
</domain>
```

Scripted properties: Scripted properties are scripts which generate properties. Properties are generated with the create methods. Every created property will be automatically sent along with values without any other operation unless discarded with the discard method. These scripts can access the following methods from ctx:

Methods	Description
Value create(name, value)	Creates a property using the specified name. It is equivalent to create(name, value, null).
Value create(name, value, appendWith)	Creates a property using the specified name and append it using the specified separator. If script reference defines output mapping, the name will be transformed to the reference output name.
void discard(value)	Discards a previously created property

A scripted-property element can have a make-variable attribute. It has exactly the same meaning as in property elements.

```
<domain>
<!-- ... -->
<script id="non-empty" in="a1 a2" out="prop">
<![CDATA[
if(ctx.str(a1).length() > 0) {
ctx.create("prop", a1);
} else if(ctx.str(a2).length() > 0) {
ctx.create("prop", a2);
}
```

```
]]>
</script>
<!-- ... -->
</domain>
```

The script may be applied with the following reference:

```
<scripted-property script-id="non-empty" in="DeviceID Description" out="part" />
```

This will create the part property from the first non empty value of DeviceID or Description. If both are empty, the property will not be created.

Instance filter scripts: Filter scripts has no special functionality but must return true or false. The returned value is the value of the last script statement (there is no return keyword).

The main difference between a pre-filter-script and a post-filter-script is that a pre-filter script can't have any input because it is executed before instance attributes polling. The only contextual information it can access is the Smarts InCharge instance name (smInstance variable).

A post-filter script can override DCF instance name. This is simply done overwritting the apgInstancevariable (which normally contains the previously set DCF instance name).

```
<domain>
<!-- ... -->
<script id="physical-device">
<![CDATA[
smInstance.indexOf("physical") >= 0
]]>
</script>
<script id="fast-eth-iface" in="Type MaxSpeed">
<![CDATA[
apgInstance = "FAST-ETH-" + apgInstance;
ctx.num(MaxSpeed) >= 100000000
&& ctx.str(Type).equals("ETHERNETCSMACD")
]]>
</script>
<!-- ... -->
<data class="Interface">
<pre-filter-script script-id="physical-device" />
<post-filter-script script-id="fast-eth-iface" in="Type MaxSpeed" />
<property name="Name" rename="part"/>
<value name="Status" rename=" Availability " unit="%">
<transformation match="UP" replace="100.0" />
<transformation match="DOWN" replace="0.0" />
</value>
</data>
<!-- ... -->
</domain>
```

This is a pretty complete example where we first filter interfaces which instance name contains the string physical and then which Type is ETHERNETCSMACD and MaxSpeed _ 100 Mbits.

Configuration hints

- Polling period: Check your polling settings to be consistent with collector settings. Indeed, the collector will retrieve data from Smarts InCharge every 4 minutes (this is the default settings, you can change it afterward) but if Smarts InCharge polling is set to an higher value, you will have some duplicated entries in the database. To change this setting, go to the Polling and Threshold panel and set every polling period according to your desired accuracy.

Logging

All DCF logs (with collector and connector logs) for a particular management pack will be found in below location:

`<DCF Install directory>/Collector-Manager/smarts-metrics-collect/logs/collecting-0-0.log`

`<DCF Install directory>/Collector-Manager/smarts-metrics-collect/logs/file-collector.txt`

Sample JSON format outputs to Kafka are below:

Sample 1:


```
{
    "groupName": "group",
    "discoveryID": null,
    "jobID": "9999",
    "type": "Interface",
    "timestamp": 1551955631,
    "value": 0.0,
    "action": "r",
    "properties": {
        "ismanaged": "true",
        "source": "INCHARGE-AM-PM",
        "type": "Interface",
        "datagrp": "SMARTS-PM-TOPO"
    },
    "metrics": {
        "ifSpeed": {
            "properties": {
                "name": "ifSpeed",
                "unit": "bit/s"
            },
            "value": 8.0E9
        }
    },
    "relations": [
        {
            "type": "Router",
            "element": "bq-gwhsrp.lss.com",
            "relationName": "PartOf"
        },
        {
            "type": "IPv6",
```

```
            "element": "IPv6-0:80C0:22C:330:0:0:0:1",
            "relationName": "Underlying"
        },
        {
            "type": "IPv6",
            "element": "IPv6-FFE:C0:22C:220:0:0:0:1",
            "relationName": "Underlying"
        },
        {
            "type": "",
            "element": "",
            "relationName": "Peer"
        }
    ],
    "initialized": true,
    "forceRefresh": true,
    "name": "IF-bq-gwhsrp.lss.com/30"
}
```


Sample 2:


```
{
    "groupName": "group",
    "discoveryID": null,
    "jobID": "9999",
    "type": "Interface",
    "timestamp": 1551955631,
    "value": 0.0,
    "action": "r",
    "properties": {
        "ismanaged": "true",
        "source": "INCHARGE-AM-PM",
        "type": "Interface",
        "datagrp": "SMARTS-PM-METRICS"
    },
    "metrics": {
        "ifInDiscards": {
            "properties": {
                "name": "ifInDiscards",
                "unit": "Pkts/s"
            },
            "value": 0.0
        },
        "ifInUcastPkts": {
            "properties": {
                "name": "ifInUcastPkts",
                "unit": "Pkts/s"
            },
            "value": 0.0
        },
        "ifOutUcastPkts": {
            "properties": {
                "name": "ifOutUcastPkts",
                "unit": "Pkts/s"
```

```
            },
            "value": 0.0
        },
        "ifOutErrors": {
            "properties": {
                "name": "ifOutErrors",
                "unit": "Pkts/s"
            },
            "value": 0.0
        },
        "ifInNUcastPkts": {
            "properties": {
                "name": "ifInNUcastPkts",
                "unit": "Pkts/s"
            },
            "value": 0.0
        },
        "ifInErrors": {
            "properties": {
                "name": "ifInErrors",
                "unit": "Pkts/s"
            },
            "value": 0.0
        },
        "ifOutNUcastPkts": {
            "properties": {
                "name": "ifOutNUcastPkts",
                "unit": "Pkts/s"
            },
            "value": 0.0
        },
        "ifOutDiscards": {
            "properties": {
                "name": "ifOutDiscards",
                "unit": "Pkts/s"
            },
            "value": 0.0
        }
    },
    "relations": [],
    "initialized": true,
    "forceRefresh": true,
    "name": "IF-bq-gwhsrp.lss.com/50"
}
```

Smarts Listener

7

This chapter includes the following topics:

- Overview
- Installation
- Configuration
- Logging

Overview

The SMARTS Event Listener is a Processing Element designed to register with the VMware SMARTS Domain managers in order to be notified whenever new notifications are received. It is fully compatible with versions 6.x, 7.x and 8.x versions of the SMARTS SAM.

The SMARTS Event Listener can only be used within an instance of Event Processing Manager. Thus, to be able to use this Processing Element, you must have a working Event Processing Manager ready.

Installation

Smarts listener component is installed as part of Smarts events SP (smarts-notifs-events) installation and hence there is no need install this separately.

Configuration

1 Event Processing Manager Configuration: To use SMARTS Event Listener instance, follow the Event Processing Manager steps.

Configuration file example:

```
File: <DCF-Install>/Event-Processing/Event-Processing-Manager/<INSTANCE>(smarts-notifs-
events)/conf/processing.xml
```

```
<processing-manager xmlns="http://www.watch4net.com/Events/DefaultProcessingManager"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.watch4net.com/Events/DefaultProcessingManager
DefaultProcessingManager.xsd ">
        <processing-element name="KAFKA" enabled="true" type="Kafka-Event-Adapter" config="Kafka-
Event-Adapter/smarts-notifs-events/conf/kafka-event-adapter.xml"/>
        <processing-element name="Smarts" enabled="true" config="Smarts-Listener/smarts-notifs-
events/conf/smarts-listener.xml" data="KAFKA"/>
        <processing-element name="EVENT-SPY" enabled="true" type="EventSpy" config="Event-
Processing-Utils/smarts-notifs-events/conf"/>
</processing-manager>
```

KAFKA processing element deals with publishing the event data from Smarts Processing element in to the Kafka.

EVENT-SPY processing element helps in debugging the event output from the logs. Event debugging can be enabled by altering the above configuration data tag like below:

```
<processing-element name="Smarts" enabled="true" config="Smarts-Listener/smarts-notifs-
events/conf/smarts-listener.xml" data="KAFKA EVENT-SPY"/>
```

In above configuration, the SMARTS Event Listener forwards events to a Kafka broker and also EVENT-SPY. Since it exclusively listens to events coming from the SMARTS domain managers like SAM, it accepts no input streams.

2 SMARTS Event Listener Configuration: The following example is the default SMARTS Event Listener configuration:

`File: <DCF-Install>/Event-Processing/Smarts-Listener//<INSTANCE>(smarts-notifs-events)/conf/smarts-listener.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://www.watch4net.com/Events/SmartsEventListener"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                           xsi:schemaLocation="http://www.watch4net.com/Events/SmartsEventListener
smarts-listener.xsd ">
        <resync-period>24h</resync-period>
        <connection-check-period>30s</connection-check-period>
        <connection-timeout>60s</connection-timeout>
        <idle-timeout>240h</idle-timeout>
```

```
        <source id="INCHARGE-SA-PRES">
                <primary id="primary">
                        <broker-hostname>localhost</broker-hostname>
                        <broker-port>4999</broker-port>
                        <domain-name>INCHARGE-SA-PRES</domain-name>
                        <username>admin</username>

<password>{1BD4D26A81F980A80617601D0EAC255B85C79E7B064E2672F0CBF9EE8BC251A6D2F68C2751691B568BF7D00D
B41E7C61}</password>
                        <notification-list>ICS_NL-ALL_NOTIFICATIONS</notification-list>
                </primary>
        </source>
</configuration>
```

resync-period: Period at which the SMARTS Event Listener will initiate a resync with the SMARTS SAM. A resync operation synchronizes data between the SMARTS Event Listener and the notifications that are currently displayed in the SMARTS SAM console. Setting this value to 0 will disable automatic resynchronization.

connection-check-period: Period at which the SMARTS Event Listener will check to make sure that its connection with the SMARTS SAM is still valid.

connection-timeout: Timeout value when trying to establish a connection with the SMARTS SAM.

idle-timeout: If no new notification is received from the SMARTS SAM after that amount of time, the SMARTS Event Listener will disconnect then reconnect to the SMARTS. This checkup is performed at the same time as the connection check. Therefore, the value of this parameter should always be equal or greater than the connection-check-period.

source: A source represents a SMARTS SAM instance to monitor. Each source can be composed of a primary source and multiple failover sources, if needed. This is useful for specifying backup SMARTS SAMs in case the primary is down.

a **primary/failover**: Determines if the source is the primary source or just a failover source.

b **broker-hostname**: The broker hostname of the SMARTS SAM if connecting to the broker as your entry point.

c **broker-port**: The port on which the SMARTS SAM's broker is accepting connections.

d *** broker-username (optional)**: The username to use to establish the connection with the broker. This password can be in the encrypted form, generated by the crypt-password script. If no authentication is required, omit this element.

e **broker-password (optional)**: The password to use to establish the connection with the broker. This password can be in the encrypted form, generated by the crypt-password script. If no authentication is required, omit this element.

f **hostname**: The hostname of the SMARTS SAM if you need to directly connect to the manager.

g **port**: The port on which the SMARTS SAM is accepting connections.

h **domain-name**: The name of the domain from which notifications will be fetched.

i **username**: The username used to connect to the domain manager.

j **password**: The password used to connect to the domain manager. This password can be in the encrypted form, generated by the crypt-password script.

k **notification-list**: This tag can be repeated as many times as necessary (one line per monitored notification list). It must be present at least once for primary sources, but it is optional for failover sources. If it is committed for a failover source, the notification lists of the corresponding primary source will be used instead.

Multiple Sources The SMARTS Event Listener can be configured to listen to many sources simultaneously. This is done by adding more source tags in the configuration file. Each source must have its primary source and can have one or more failover sources.

3 Kafka Event Adapter Configuration: The Kafka Event Adapter is a component that is able to both read events from Kafka and write events to Kafka.

Kafka server configuration used to publish events from SAM is below:

File: `<DCF-Install>/Event-Processing/Kafka-Event-Adapter/<INSTANCE>(smarts-notifs-events)/conf/kafka-event-adapter.xml)`

```
<?xml version="1.0"?>
<kafka-event-adapter-config xmlns="http://www.watch4net.com/KafkaEventAdapter"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.watch4net.com/KafkaEventAdapter ../kafka-event-adapter.xsd ">
  <cluster>
    <server host="localhost" port="9092"/>
    <additional-parameters key="security.protocol">SSL</additional-parameters>
        <additional-parameters key="ssl.truststore.location">../../../Tools/Webservice-
Gateway/Default/conf/truststore</additional-parameters>
    <additional-parameters
key="ssl.truststore.password">{F871B10293EEB1C941E2EA5466F817546662FD1314591713B73E73A7E39663A99602
55C4B844F536409BD410490E007F}</additional-parameters>
    <additional-parameters key="ssl.keystore.location">../../../Tools/Webservice-
Gateway/Default/conf/clientkeystore</additional-parameters>
    <additional-parameters
key="ssl.keystore.password">{85DF870D632462AF411DB8164B9455741BCCCB1CE493C475B7C121E5CEFA2189A2CDE6
CE65466BE4C2E99175CAEFA6F1}</additional-parameters>
    <additional-parameters
key="ssl.key.password">{DB17AC06BD3C6420FAA350241DFC43BE4504E0173A8BC2FA0C7FC9D79892374392195CD3EB1
5BD3D2D914FD470BF7075}</additional-parameters>
  </cluster>

    <event-writer topic-name="default-topic" stream="data" isJson="true">
        <kafka-producer-settings>
            <producer compression-type="none" />
        </kafka-producer-settings>
    </event-writer>

</kafka-event-adapter-config>
```

Kafka Event Adapter parameters

Parameter	Description
<cluster>	This tag must occur at least once.
<server>	This tag must occur at least once: * **host**: The address of one of the kafka bootstrap servers * **port**: The port that the bootstrap server is listening on
<producer>	This tag is optional, but may be used for templating kafka producers. Refer to the schema for more information.
<consumer>	This tag is optional, but may be used for templating kafka consumers. Refer to the schema for more information.
<connection>	This tag is optional, but may be used for templating kafka connections. Refer to the schema for more information.
<additional-parameters>	This tag is optional, but may be used for configuring kafka options outside of the purview of the previous tags. These options include, but are not limited to SSL connection parameters.

Configuring Event Writer parameters

Parameter	Description
<event-writer>	This tag is used to define a component that will write to kafka **topic-name** : The topic user will write to. **stream** : The event stream to consume from. **isJson**: Json format output is enabled
<kafka-producer-settings>	This tag may be used if we want to customize how the writer, writes to kafka. Refer to the schema for more information.
<connector-component-behavior>	This tag may be used if user want to control how often user flush to kafka.
<key-encoder>	This tag may be used if user want to customize how the kafka key is encoded.
<value-encoder>	This tag may be used if user want to customize how the kafka value is encoded.

Configuring Event Reader parameters

Parameter	Description
<event-reader>	This tag is used to define a component that will consumer from kafka. **topic-name** : This attribute is used to specify the kafka topic to read from. **stream** : This attribute is used to specify the stream to push the events to once they have been read
<kafka-consumer-task>	This tag may be used to customize the way that we will read from kafka. **poll-timeout-ms** : The poll timeout for kafka. **error-timeout-ms** : The error timeout for connecting to kafka. **error-timout-double-n** : How many errors before we double the timeout. **error-timout-max-ms** : The maximum error timeout.
<consumer>	This tag may be used to customize how this element will consume from kafka. Of note, is that the attribute group-id must be used here if we plan on joining a kafka consumer group.
<connection>	This tag may be used to customize the kafka connection for this element.
<additional-parameters>	This tag may be used to customize the additional-parameters for this element.
<initial-topic-partition-offset-seeker>	This element is used when wanting to control how we seek in a kafka topic. **existing-reset-policy** : policy when seeking on existing partition but the requested offset no longer exists. Must be one of earliest or latest. **new-reset-policy** : policy when seeking on a new partition. Must be one of earliest or latest.
<topic-partition-offset-commit>	This element must be used if we would like to control how our offsets are committed. **location** : where to commit the topic partition offsets. Must be one of disk, kafka, none. **commit-interval** : Offsets are committed no more frequently than this interval.
<topic-partition-listener>	Required if kafka-consumer group is not set. Used to refresh the list of topic partitions periodically. **new-reset-policy** : where to seek in new partitions. (at earliest or latest offset in the new partition: usually at the earliest). **existing-reset-policy** : where to seek in existing partitions when current offset is out of range. (usually at the earliest). **refresh-interval** : how often to check for new partitions.

Data Processing

If user want to alter default event data values before pushing to Kafka, they can do so by using below event processers. Each of these processors usage is explained in respective sections:

1 Event Log processor

2 Event Property Tagger

Logging

All Event processing logs for a particular management pack can be found at: `<DCF-Install>/Event-Processing/Event-Processing-Manager/<INSTANCE>(ex:smarts-notifs-events)/logs`.

Event processing log can be found in below path. This will have event spy logging info also, if enabled.

`<DCF-Install>/Event-Processing/Event-Processing-Manager/<INSTANCE>(ex:smarts-notifs-events)/logs/processing-0-0.log`

Latest processing log name will have "-0-0" appended to its name like above.

To enable logging, change the configuration to push events to event spy. And after changing the configuration, operator need to restart the service to see the logging info in processing-0-0.log.

Example: *smarts-notifs-events* is installed management pack name in below example:

To check the status of the installed services:

```
<DCF-Install>/bin/manage-modules.sh service status all
….
* Checking 'event-processing-manager smarts-notifs-events'...       [ running ]

To restart the particular service:

<DCF-Install>/bin/manage-modules.sh service restart event-processing-manager smarts-notifs-events
 * Stopping 'event-processing-manager smarts-notifs-events'...            [ OK ]
 * Starting 'event-processing-manager smarts-notifs-events'...            [ OK ]
```

Sample output (JSON) of event in Kafka :


```
{
    "Acknowledged": false,
    "Active": true,
    "AuditTrail": [
        {
            "User": "admin",
            "ActionType": "UNACKNOWLEDGE",
            "SerialNumber": 3,
            "Text": "",
            "Timestamp": 1550654825
        },
        {
            "User": "admin",
            "ActionType": "RELEASE_OWNERSHIP",
            "SerialNumber": 2,
```

```
            "Text": "",
            "Timestamp": 1550654778
        },
        {
            "User": "admin",
            "ActionType": "ACKNOWLEDGE ALL",
            "SerialNumber": 1,
            "Text": "",
            "Timestamp": 1550654728
        },
        {
            "User": "DXA",
            "ActionType": "NOTIFY",
            "SerialNumber": 0,
            "Text": "Server: INCHARGE-AM-PM",
            "Timestamp": 1550654481
        }
    ],
    "Category": "Availability",
    "CausedBy": [],
    "Causes": [
        "NOTIFICATION-Router_10.107.116.235_Unresponsive",
        "NOTIFICATION-Router_200.0.4.1_Unresponsive",
        "NOTIFICATION-HSRPGroup_HSRP-GROUP-10.10.20.5_AllComponentsDown"
    ],
    "Certainty": 100.0,
    "ClassDisplayName": "Router",
    "ClassName": "Router",
    "ClearOnAcknowledge": false,
    "ElementClassName": "Router",
    "ElementName": "10.107.116.235",
    "EventDisplayName": "Down",
    "EventName": "Down",
    "EventState": "ACTIVE",
    "EventText": "Indicates that the root cause is a failed system.",
    "EventType": "DURABLE",
    "FirstNotifiedAt": 1550654416,
    "Impact": 0,
    "InMaintenance": false,
    "InstanceDisplayName": "10.107.116.235",
    "InstanceName": "10.107.116.235",
    "IsProblem": true,
    "IsRoot": true,
    "LastChangedAt": 1550654916,
    "Name": "NOTIFICATION-Router_10.107.116.235_Down",
    "OccurrenceCount": 1,
    "Owner": "admin",
    "PollingState": "SUBSCRIPTION",
    "ProcessedTimeStamp": 1550654916,
    "Severity": 1,
    "Source": "INCHARGE-SA-PRES",
    "SourceDomainName": "INCHARGE-AM-PM",
    "SourceEventType": "PROBLEM",
    "SourceInfo": "",
    "TroubleTicketID": "",
```

```
    "UserDefined1": "",
    "UserDefined10": "",
    "UserDefined11": "",
    "UserDefined12": "",
    "UserDefined13": "",
    "UserDefined14": "",
    "UserDefined15": "",
    "UserDefined16": "",
    "UserDefined17": "",
    "UserDefined18": "",
    "UserDefined19": "",
    "UserDefined2": "",
    "UserDefined20": "",
    "UserDefined3": "",
    "UserDefined4": "",
    "UserDefined5": "",
    "UserDefined6": "",
    "UserDefined7": "",
    "UserDefined8": "",
    "UserDefined9": ""
}
```

Sample debugging (Event-Spy) output from processing-0-0.log

```
INFO     -- [2019-03-13 16:58:14 IST] -- EventSpy$SpyStreamHandler::handleEvent(): From Smarts[data]:
INFO     -- [2019-03-13 16:58:14 IST] -- EventSpy$SpyStreamHandler::handleEvent():
com.watch4net.events.common.data.GenericEvent
(k) Name                = NOTIFICATION-Router_bq-
gwlab.lss.com_DiscoveryError

                                                                                    (DEFINITION,STRING)
    ClassName           =
Router


 (DEFINITION,STRING)
    InstanceName        = bq-
gwlab.lss.com


(DEFINITION,STRING)
    EventName           =
DiscoveryError


 (DEFINITION,STRING)
```