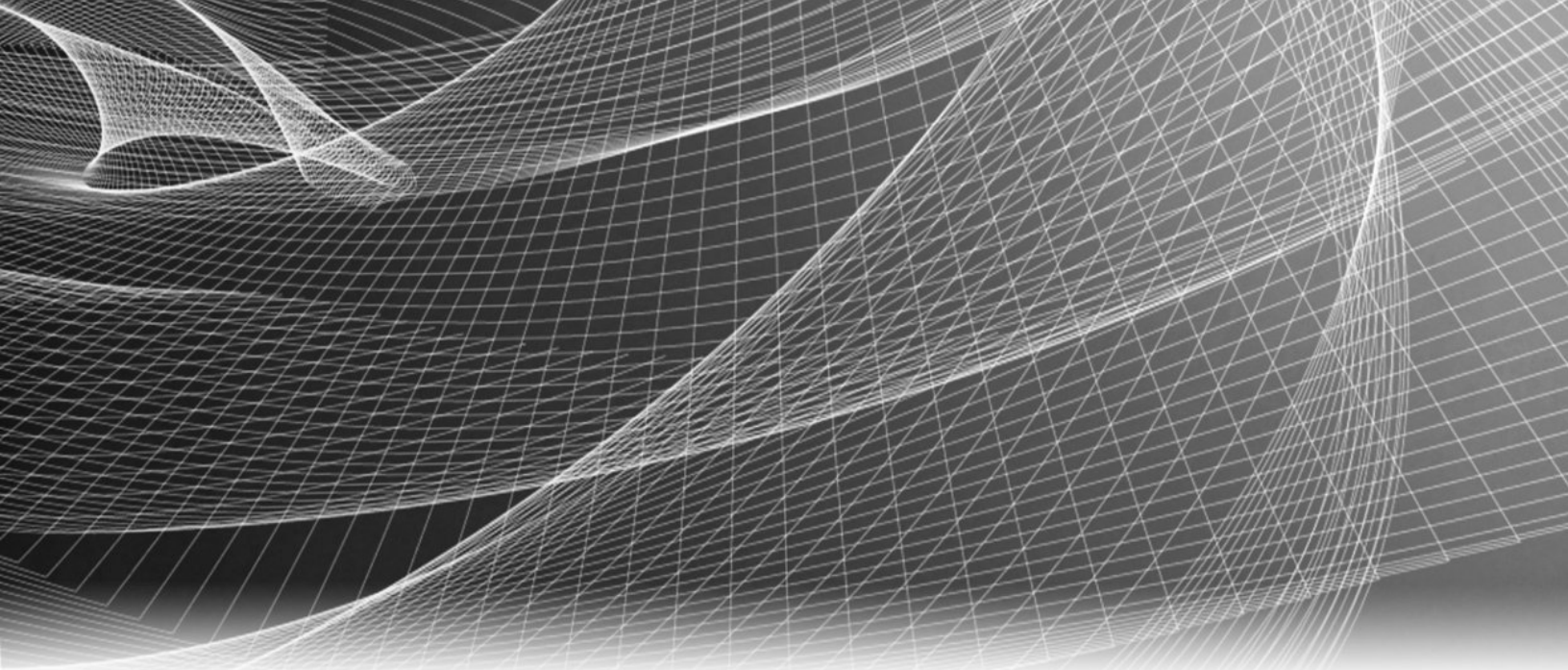




EMC® Watch4net®

Version 6.3u2

Event Log Processor Admin Guide

P/N: pending

Rev : 40032

Copyright ©2010 - 2014 EMC Corporation. All rights reserved. Published in the USA.

Published April, 2014

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC², EMC, and the EMC logo are registered trademarks or trademarks of EMC Corporation in the United States and other countries. All other trademarks used herein are the property of their respective owners.

For the most up-to-date regulatory document for your product line, go to EMC Online Support (<https://support.emc.com>).

Contents

Preface	iii
Purpose	iii
Intended Audience	iii
Prerequisites	iii
Document Organization	iii
Where to get help	v
1 Architectural Overview	1
2 Configuration	3
1 APG Event Processing Manager Configuration	3
2 APG Event Log Processor Configuration	4
3 Rule Reference	5
1 Basic Rules	5
1.1 Rule Chain (rules)	5
1.2 Include Rule (include)	5
1.3 Switch Rule (switch)	5
1.4 Forward Rule (forward)	6
1.5 Noop Rule (noop)	6
1.6 Log Rule (log)	6
1.7 Count Rule (count)	7
1.8 Copy Rule (copy)	7
1.9 Set Rule (set)	7
2 Tagging Rules	7
2.1 Timestamp Rule (timestamp)	7
2.2 UUID Rule (uuid)	7
2.3 Sequence Rule (seq)	8
2.4 Id Generator Rule (id)	8
2.5 Link Rule (link)	8
2.6 MAC Rule (mac)	8
2.7 Format Rule (format)	8
3 Matching Rules	9
3.1 Exists Rule (exists)	9
3.2 Starts With Rule (starts-with)	9
3.3 Contains Rule (contains)	9
3.4 Equals Rule (equals)	9
3.5 Not Equals Rule (not-equals)	10
3.6 Regex Rule (regex)	10
4 Parsing Rules	10
4.1 Date Parser Rule (parse-date)	10
4.2 Duration Parser Rule (parse-duration)	11
4.3 Syslog <i>Priority</i> Parser Rule (parse-pri)	11
4.4 IPv4 To Integer Rule (ipv4-to-int)	11
4.5 Integer To IPv4 Rule (int-to-ipv4)	11

5	Advanced Rules	11
5.1	Tokenizer Rule (tokenize)	11
5.2	Custom Rule (custom)	12

Preface

Purpose

The *APG Event Log Processor* is a powerful *Processing Element* that can filter and transform incoming event streams using a set of rules. It can be used in many different contexts such as log management, data enrichment, etc.

Intended Audience

This guide is addressed to the *APG* system administrator that wants to configure the *APG Event Log Processor*.

Prerequisites

The *APG Event Log Processor* runs in an instance of *APG Event Processing Manager*. It requires the *APG Event Processing Manager* to be installed and configured correctly. Some background knowledge of the *APG* event framework is desirable to understand some key *APG Event Processing Manager* concepts.

Document Organization

1. Architectural Overview	This section describes the <i>APG Event Log Processor</i> and its basic functionality.
2. Configuration	This section explains how to configure the <i>APG Event Log Processor</i> .
3. Rule Reference	This section describes all the available rules you can use with the <i>APG Event Log Processor</i> .

Where to get help

EMC support, product, and licensing information can be obtained as follows:

PRODUCT INFORMATION - For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

<https://support.emc.com>

TECHNICAL SUPPORT - Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

Architectural Overview

The APG *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.

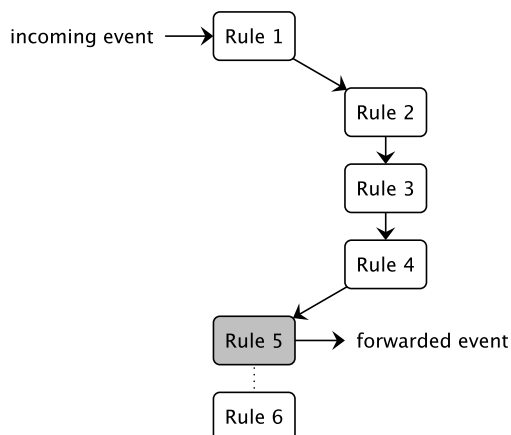


Figure 1: Rule Evaluation Example

In the previous example, 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.



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](#) on page 1, 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 [APG Event Log Processor Configuration](#) on page 4 for an example of the override syntax.

Configuration

1 APG Event Processing Manager Configuration

In order to use the *APG Event Log Processor*, you have to declare it in the *APG Event Processing Manager* configuration (typically `APG/Event-Processing/Event-Processing-Manager/Default/conf/processing.xml`).

```
<?xml version="1.0" encoding="UTF-8"?>
<processing-manager xmlns="http://www.watch4net.com/Events/DefaultProcessingManager">

  <!-- ... -->

  <processing-element name="LOG" config="Event-Log-Processor/Default/conf/config.xml" />

  <!-- ... -->

</processing-manager>
```

The previous declaration would create an *APG Event Log Processor* using `conf/config.xml` as its configuration file. For the simplicity of this example, no input or output for the *APG Event Log Processor* is declared.

The *APG Event Log Processor* accepts any input stream (incoming events on any stream will be forwarded to the root rule indifferently) but only allows output streams that have been declared in its rule set. For example, to forward data to another *Processing Element* through the data stream, you would have to declare a forward rule configured to use the data stream first.



A complete example is available in `misc/processing.xml`. It shows a typical *APG Event Processing Manager* configuration which uses the *APG Event Log Processor* along with a listener and a database writer.

2 APG Event Log Processor Configuration

The APG *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 this 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 complete example is available in `misc/event-log.xml`. It shows multiple examples of commonly used rules in order to parse raw events.

Rule Reference

1 Basic Rules

1.1 Rule Chain (rules)

The rule chain is the core of the APG *Event Log Processor*. It basically evaluates a list of rules as described in section 1: [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>
```

1.2 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" />
```

1.3 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 doesn't 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` $\neq$ `42` $\neq$ `"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>
```

1.4 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 APG *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" />
```

1.5 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" />
```

1.6 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 APG *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" />
```

1.7 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" />
```

1.8 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" />
```

1.9 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" />
```

2 Tagging Rules

2.1 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" />
```

2.2 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" />
```

2.3 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" />
```

2.4 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" />
```

2.5 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" />
```

2.6 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 *APG's* `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>
```

2.7 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, please refer to <http://download.oracle.com/javase/6/docs/api/java/text/MessageFormat.html>.
- `printf` uses a *Java Formatter* which format string syntax is similar to the well-known *C printf*. For more information on the format string syntax, please refer to <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 *placeholder* 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>
```

3 Matching Rules

3.1 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" />
```

3.2 Starts With Rule (`starts-with`)

This rule returns `continue` if a specific field (designated by the `field` attribute) starts 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" />
```

3.3 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" />
```

3.4 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` $\neq$ `42` $\neq$ `"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" />
```

3.5 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" />
```

3.6 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, please refer to 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:_">
```

4 Parsing Rules

4.1 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. The pattern must be compatible with the standard *Java* data parser (see the documentation [here](#)) and 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="MMMdd_yyyy_HH:mm:ss" />
<parse-date name="Parses_a_date_in_GMT" field="Date" to="Timestamp" timezone="GMT" pattern="MMMdd_yyyy_HH:mm:ss" />
```

4.2 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 `continue` if the duration field has been successfully parsed, `failure` if not.

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

4.3 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" />
```

4.4 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" />
```

4.5 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 omitted, `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" />
```

5 Advanced Rules

5.1 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>
```

5.2 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 *APG Event Log Processor*.

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