



EMC® Watch4net®

Version 6.3u2

Event Key Modifier Processor Admin Guide

P/N: pending

Rev : 40032

EMC²

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

Published April, 2014

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

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

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

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

Contents

Preface	iii
Purpose	iii
Intended Audience	iii
Prerequisites	iii
Document Organization	iv
Where to get help	v
1 Overview	1
1 Product Overview	1
2 Configuration Overview	2
2 Configuration Ordering	5
3 <i>Event Processing Manager</i> Configuration	9
4 <i>Event Key Modifier</i> Configuration	11
5 Factories	13
1 EnforcedValidityEventFactory	13
2 NoKeyValidationEventFactory	14
3 NoKeyValidationEventFactory	14
6 Appendix A: XML Schema	15
7 Appendix B: Using an EventSpy	17

Preface

Purpose

The *Event Key Modifier* is a *Processing Element* designed to change the fields that comprise the key for an event.

Intended Audience

This guide is intended for advanced *APG* Administrators that have a good knowledge of the *APG Event Processing* framework.

Prerequisites

The *Event Key Modifier* can only be used within a working instance of the *Event Processing Manager*, version 5.1 or above.

Document Organization

1. Overview	Provides a quick overview of this product, as well as configuration overview steps.
2. Configuration Ordering	Describes and details the overall workflow necessary when modifying keys for events where an SQL table and database schema have not been pre-defined.
3. <i>Event Processing Manager</i> Configuration	How to configure the <i>Event Processing Manager</i> to include an <i>Event Key Modifier</i> .
4. <i>Event Key Modifier</i> Configuration	How to configure the <i>Event Key Modifier</i> itself.
5. Factories	Available factories for the <i>Event Key Modifier</i> to output events with new key fields.
Appendix A: XML Schema	The XML Schema that the <i>Event Key Modifier</i> configuration file is based on, for reference.
Appendix B: Using an EventSpy	Using an EventSpy to see an event's field names and values, field types, and current key composition. This information can be used to compose a new key.

Where to get help

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

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

<https://support.emc.com>

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

1

Overview

The following section provides an overview of the product and its configuration.

1 Product Overview

The *Event Key Modifier* replaces which fields in an event together compose the event's key. All events that pass through the *Event Key Modifier* are tagged with the same key.

Event keys are used to identify individual events that will be written to a table. So, by using the *Event Key Modifier* to tag all incoming events with the same key, all of the output events will overwrite each other producing a single record in the table with the last received event using this newly defined key. This is a method of grouping events that would have otherwise been written individually to the table with no apparent relationship.

The Event Key Modifier can also validate whether new key fields already exist in the incoming event, and whether they are of the correct type: `OCCURRENCE` or `DEFINITION`. Fields used for the new key must each already exist, be of the same type as indicated in the incoming event and in the same order as defined in the endpoint database, to create a valid output event that

will be written to the database. See [Configuration Ordering](#) on page 5 for an example.

The Event Key Modifier currently only works with the MySQL database using the Generic-Live-Writer. It is not relevant when using the Datastore since keys are not used in this case.

2 Configuration Overview

Follow the workflow steps below to configure the *Event Key Modifier* and its complement modules:

1. If necessary, perform the following preparatory steps:
 - Create a new table in the SQL database that has all of the necessary fields in the desired order, along with a primary key with correspondingly ordered key fields. The fields, and their order, must match the key fields you will define the *Event Key Modifier* to apply to incoming events.
 - Create a new database writer mapping configuration file to include the fields in the same order and identified as regular, or key, fields. For both of these steps, see [Configuration Ordering](#) on page 5.

Note:

The above preparatory steps might not be necessary, for example, if the *Event Key Modifier* is a part of a ReportPack where necessary tables and schemas have been predefined as part of its implementation. See the ReportPack documentation for more information.

2. Configure the **processing.xml** file for the *Event Processing Manager* to recognize one or more *Event Key Modifiers*. See [Event Processing Manager Configuration](#) on page 9.

Since all events that pass through the *Event Key Modifier* are tagged with the same key, you need to configure additional *Event Key Modifier* instances in different flows, if you want to use additional keys for other groups of events.

3. Create one or more *Event Key Modifier* configuration files depending on your number of *Event Key Modifier* instances. See [Event Key Modifier Configuration](#) on page 11.

To help you in your configuration, you could do the following:

- Reference the XML Schema the configuration file is based on. See [Appendix A: XML Schema](#) on page 15.

-
- Reference definitions of the available factories which either use or don't use validation when generating new keys for events. See [Factories](#) on 13.
 - Advanced users can use the `EventSpy` to generate logs to see information about incoming events, their fields, values and field types, which they could then use in the configuration. See [Appendix B: Using an EventSpy](#) on 17.
4. Configure other elements in the event flow that will send events as input to the *Event Key Modifier*. See module documentation.
 5. Configure other elements in the event flow that will receive the event output from the *Event Key Modifier*. See module documentation.
 6. Restart the *Event Processing Manager*.

2

Configuration Ordering

When using the *Event Key Modifier* to change a key, it must match the primary key of a table to be written to the database.

If you are customizing the *Event Key Modifier* for your particular purposes, you will have to create a new table with the correct fields in a particular order which both the primary key uses, and which will determine the key order when configuring the *Event Key Modifier* configuration file to apply to incoming events. You will also have to create a new Generic-Live-Writer mapping configuration, again, with the same key order and key fields, to be able to write events with the new key to the SQL database.

You can use an EventSpy to see the fields in an incoming event so you can select from those which ones you want to preserve in the database and which ones to use as key fields. For information on setting up and using an EventSpy, see [Appendix B: Using an EventSpy](#) on 17.

An example output from the EventSpy would list field names, their values, whether they are of type OCCURRENCE or DEFINITION, and what is their data type:

```
fieldA = value for A (DEFINITION, LONG)
(k) fieldB = value for B (OCCURRENCE, STRING)
fieldC = value for C (OCCURRENCE, LONG)
(k) fieldD = value for D (OCCURRENCE, LONG)
fieldE = value for E (OCCURRENCE, STRING)
```

The fields preceded by a (k) are the current key fields in the event.

You would then create a table with the fields in the desired order, and with a primary key that uses the same order for its fields—the primary key will use a subset of the available fields which are distributed among all the fields with the overall order being consistent.

So, following the example EventSpy output above, indicating available fields in an incoming event, you might create a new table in the MySQL database, with a new primary key with the following SQL statement:

```
create table event_modified1 (
    fieldE varchar(255),
    fieldD int(11),
    fieldC int(11),
    fieldB varchar(255),
    fieldA int(11),
    primary key(fieldE, fieldC, fieldA)
);
```

As you can see, the fields have been rearranged in a different order, and the primary key for the table is created from the existing fields which are in the same order as they appear in the overall field list.

Then, as follows, in the *Event Key Modifier* configuration you also need to specify the new key definition in the same order as the primary key field order of the table the output events will be written to:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://www.watch4net.com/events/processing/keymodifier"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.watch4net.com/events/processing/keymodifier_event-
key-modifier.xsd">

    <factory>com.watch4net.events.processing.keymodifier.factories.
EnforcedValidityEventFactory</factory>
    <key>
        <field name="fieldE" type="OCCURENCE"/>
        <field name="fieldC" type="OCCURENCE"/>
        <field name="fieldA" type="DEFINITION"/>
    </key>
</configuration>
```

For full information on the *Event Key Modifier* configuration file, see [Event Key Modifier Configuration](#) on page 11.

You then must create a new database writer mapping file for the Generic-Live-Writer and then reference this in the **processing.xml** file.

To help you in its configuration, you can use the file **my-smarts-archive.xml** as a basis by renaming it and then proceeding to adjust its content. You can also reference the schema file the mapping is based on, called **GenericLiveWriter.xsd**. Both of these files are located at:

```
<APG>/Event-Processing/Generic-Live-Writer/<instance name>/conf
```

For this example, let's say we renamed the mapping file to **key-modifier-mapping1.xml**. It can be renamed to anything as long as it is referenced properly.

Then, in the **key-modifier-mapping1.xml**, you would change the value of the table attribute to the name of the new table you created above, like the following:

```
<personality xsi:type="mysql" table="event_modified1" />
```

Then proceed to create the mapping for our example, as follows:

```
<mapping>
  <key field="fieldE" field-type="OCCURRENCE" data-type="STRING" xsi:type="
DefaultFieldMapping"/>
  <field field="fieldD" field-type="OCCURRENCE" data-type="LONG" xsi:type="
DefaultFieldMapping"/>
  <field field="fieldC" field-type="OCCURRENCE" data-type="LONG" xsi:type="
DefaultFieldMapping"/>
  <key field="fieldA" field-type="DEFINITION" data-type="LONG" xsi:type="
DefaultFieldMapping"/>
  <key field="fieldB" field-type="OCCURRENCE" data-type="STRING" xsi:type="
DefaultFieldMapping"/>
</mapping>
```

Finally you would make sure the **processing.xml** file points to the GENERIC-LIVE-WRITER configuration file you created where the *Event Key Modifier* output will be written to the database:

```
<!-- a database writer -->
<processing-element name="writer" config="Generic-Live-Writer/Default/conf/key-modifier-
mapping1.xml" />
```

For full information on the **processing.xml** file, see [Event Processing Manager Configuration](#) on page 9.

Note: If you are using multiple instances of the *Event Key Modifier*, you would perform the steps in this section, creating a new table, configuring the *Event Key Modifier* instance, the database writer, and the *Event Processing Manager*, additional times.

3

Event Processing Manager Configuration

To use an *Event Key Modifier* instance you must modify the *Event Processing Manager*'s **processing.xml** file to include a line for each *Event Key Modifier* you want to configure.

The path to the file is:

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

Please follow the example below to configure the *Event Processing Manager*:

```
<!-- a database writer -->  
<processing-element name="writer" config="some/writer/config" />  
  
<!-- the Event Key Modifier -->  
<processing-element name="modifier" config="Event-Key-Modifier/Default/conf/key-modifier-  
default.xml" data="writer" />
```

```
<!-- an event listener -->  
<processing-element name="listener" config="some/listener/config" data="modifier"/>
```

The *Event Key Modifier* usually forwards events to a *Database Writer*, but can forward events to other processing elements in the event flow as well. This processing element only accepts a data input stream from other processing elements in the event flow and correspondingly only supports a data output stream. Data is the default type of stream.

4

Event Key Modifier Configuration

The following example is the default *Event Key Modifier* configuration. This configuration is based on the XML schema which can be found in [Appendix A: XML Schema](#) on page 15.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://www.watch4net.com/events/processing/keymodifier"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.watch4net.com/events/processing/keymodifier_event-
key-modifier.xsd">

    <factory>com.watch4net.events.processing.keymodifier.factories.
EnforcedValidityEventFactory</factory>
    <key>
        <field name="Source" type="DEFINITION"/>
        <field name="OpenedAt" type="OCCURRENCE"/>
    </key>
</configuration>
```

- **factory** : Class used to change incoming event keys to one with the specified key fields; see

[Factories](#) on page 13 for more informations about available factories. You must know the event fields that are a part of the event to set your new key. For this purpose, you can use an `EventSpy` to track the event fields in a log file. See [Appendix B: Using an EventSpy](#) on page 17.

- `key` : Specifies the new key with the specified `fields`. This element can have 0 or more `field` elements defined with the following attributes:
 - `name` : Specifies the field name to put in the key.
 - `type` : Specifies the type of the field. Either `OCCURRENCE` or `DEFINITION`. Fields can be of either type, but they must retain the same type on output as they had on input. See [Factories](#) on page 13.

Note:

When no key fields are configured, the output events will have no defined key. This will force them to all be individually written to the database table.

Factories

The following section presents the available factories for generating new keys for events.

1 EnforcedValidityEventFactory

Complete name is:

```
com.watch4net.events.processing.keymodifier.factories.EnforcedValidityEventFactory
```

This factory uses a simple validation to make sure the fields are already added to the event before trying to set them as key fields. If the fields to be used as the new key do not exist in the incoming event, the flow fails: Events must reach each destination along a path, which might be across several processing elements, before another instance of the event is sent along the next possible path. So if the event flow fails, it will stop the event from following any further paths in the flow.

When the *Event Key Modifier* tries to add a key field, it also looks up the event to see which type, OCCURRENCE or DEFINITION, it will ascribe to the key field. It doesn't use the type attribute in the XML file, which is ignored.

Therefore, using this factory you should make sure the fields used for the new key exist in the incoming events, but you do not need to concern yourself with the event type.

2 NoKeyValidationEventFactory

Complete name is:

```
com.watch4net.events.processing.keymodifier.factories.RedefineFieldsTypeInKeyEventFactory
```

This factory allows the redefinition of the field type so one can change an occurrence field to a definition field or a definition field to an occurrence field. This factory uses the type attribute from the XML file to determine which type, OCCURRENCE or DEFINITION, to ascribe to each key field.

3 NoKeyValidationEventFactory

Complete name is:

```
com.watch4net.events.processing.keymodifier.factories.NoKeyValidationEventFactory
```

This factory makes no validation whatsoever before generating the new key for the event, therefore it should only be used by expert users who can predict the composition of the incoming events. This factory uses the type attribute from the XML file to determine which type, OCCURRENCE or DEFINITION, to ascribe to each key field.

Because no validation is used by this factory, the event will fail upon being written to the database if the fields used for its key did not exist in the incoming event or are of a different type than in the incoming event. This failure occurs because event keys are associated to primary keys defined for a table in the database. As explained above, events must reach each destination along a path, which might be across several processing elements, before another instance of the event is sent along the next possible path. So if the event, in this case, cannot be written to the database, it will stop the event from following any further paths in the flow.

By not performing the validation that occurs with the other factory, this factory is much faster which is ideal when performance is critical.

6

Appendix A: XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  targetNamespace="http://www.watch4net.com/events/processing/keymodifier"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.watch4net.com/events/processing/keymodifier"
  elementFormDefault="qualified">

  <xsd:element name="configuration" type="ProcessingConfiguration" />

  <xsd:complexType name="ProcessingConfiguration">
    <xsd:sequence>
      <xsd:element name="factory" minOccurs="1" maxOccurs="1" type="xsd:string" />
      <xsd:element name="key" minOccurs="1" maxOccurs="1" type="key" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="key">
    <xsd:sequence>
      <xsd:element name="field" type="field" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="field">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="type" type="KeyType" use="optional"/>
      </xsd:complexType>

      <xsd:simpleType name="KeyType">
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="OCCURRENCE" />
          <xsd:enumeration value="DEFINITION" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:schema>
```

Appendix B: Using an EventSpy

With the EventSpy utility, you can see all the field names, their values, type (OCCURRENCE or DEFINITION), as well as their current key.

To use an EventSpy do the following:

1. If not already done, install the package Event-Processing-Utils.
2. Add the following line to the **processing.xml** file:

```
<processing-element name="SPY" config="Event-Processing-Utils/Default/conf/" type="EventSpy" />
```

3. Make sure your flow uses the EventSpy by forwarding data output to the EventSpy input:

```
<processing-element name="listener" config="some/listener/config" data="SPY"/>  
<processing-element name="SPY" config="Event-Processing-Utils/Default/conf/" type="EventSpy" />
```

4. Restart the *Event Processing Manager*.

5. After some events have occurred, examine the log files for the *Event Processing Manager* in `<APG>/Event-Processing/Event-Processing-Manager/<instance name>/logs` for a record of the events, their keys, field names and values, and types.
6. Proceed to use the information gathered to configure the *Event Key Modifier*, indicating key fields and types, which will not be validated when using the `NoKeyValidationEventFactory` factory.