



EMC® Watch4net®

Version 6.3u2

Event Failover Administration Guide

P/N: pending

Rev : 40032

EMC²

Copyright ©2011 - 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 <i>Event Processing Manager Configuration</i>	1
1 Example with transmitters	3
2 Example with separate listeners and writers	4
2 <i>Event Failover Configuration</i>	5

Preface

Purpose

The *Event Failover* is a *Processing Element* designed to ensure that no event is lost between two other *Processing Elements*. If the receiver side is down, the sender will buffer the incoming events and keep them on disk until the responder is back.

Intended Audience

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

Prerequisites

The *Event Failover* can only be used within an instance of the *Event Processing Manager*. To be able to use this *Processing Element*, you must have a working *Event Processing Manager* ready.

Document Organization

1. <i>Event Processing Manager Configuration</i>	How to configure the <i>Event Processing Manager</i> to include a <i>Event Failover</i> .
2. <i>Event Failover Configuration</i>	How to configure the <i>Event Failover</i> itself.

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

Event Processing Manager Configuration

To use an *Event Failover* instance on the sender side, please add this line in the *Event Processing Manager* configuration file:

```
<!-- Element that's send the data on the network -->
<processing-element name="Element-That-Write-Data-On-The-Network" config="Writer/Default/conf/
test.xml" />

<!-- Element that's receive ACK from the responder -->
<processing-element name="Element-That-Read-Data-On-The-Network" config="Reader/Default/conf/
test.xml" data="Event-Failover-Buffer[command]" />

<!-- Event Failover Buffer -->
<processing-element name="Event-Failover-Buffer" config="Event-Failover/Default/conf/buffer.xml
" data="Element-That-Write-Data-On-The-Network[data]" />

<!-- Other element -->
<processing-element name="Element-That-Sends-Data" config="Test/Default/conf/test.xml" data="
```

```
Event-Failover-Buffer[data]" />
```

To use a *Event Failover* instance on the receiver side, please add this line in the *Event Processing Manager* oconfiguration file:

```
<!-- Element that's send the ACK on the network -->
<processing-element name="Element-That-Write-Data-On-The-Network" config="Writer/Default/conf/
test.xml" />

<!-- Element that's reading the data on the network -->
<processing-element name="Element-That-Read-Data-On-The-Network" config="Reader/Default/conf/
test.xml" data="Event-Failover-Responder[data]" />

<!-- Event Failover Responder -->
<processing-element name="Event-Failover-Responder" config="Event-Failover/Default/conf/
responder.xml" command="Element-That-Write-Data-On-The-Network[data]" data="Element-That-Wait-
For-Data[data]" />

<!-- Other element -->
<processing-element name="Element-That-Wait-For-Data" config="Test/Default/conf/test.xml" />
```

The *Event Failover* can receive events on as many input streams as required. Events received on each input stream will be written in a buffer and sent on to the next element that must eventually reach a responder. The command stream is used to acknowledge the reception of an event of the responder removing them from the buffer.

1 Example with transmitters

This is the suggestion configuration method. Sender side:

```
<!-- Event Transmitter -->
<processing-element name="Transmitter" config="Generic-Event-Transmitter/Default/conf/
transmitter.xml" data="Event-Failover-Buffer[command]"/>

<!-- Event Failover Buffer -->
<processing-element name="Event-Failover-Buffer" config="Event-Failover/Default/conf/buffer.xml
" data="Transmitter[data]" />

<!-- Other element -->
<processing-element name="Element-That-Sends-Data" config="Test/Default/conf/test.xml" data="
Event-Failover-Buffer[data]" />
```

Receiver side:

```
<!-- Event Transmitter -->
<processing-element name="Transmitter" config="Generic-Event-Transmitter/Default/conf/
transmitter.xml" data="Event-Failover-Responder[data]"/>

<!-- Event Failover Responder -->
<processing-element name="Event-Failover-Responder" config="Event-Failover/Default/conf/
responder.xml" command="Transmitter[data]" data="Element-That-Wait-For-Data[data]" />

<!-- Other element -->
<processing-element name="Element-That-Wait-For-Data" config="Test/Default/conf/test.xml" />
```

2 Example with separate listeners and writers

Sender side:

```
<!-- Event Writer -->
<processing-element name="Writer" config="Generic-Event-Writer/Default/conf/writer.xml" />

<!-- ACK Listener -->
<processing-element name="Listener" config="Generic-Event-Listener/Default/conf/test.xml" data=
"Event-Failover-Buffer[command]" />

<!-- Event Failover Buffer -->
<processing-element name="Event-Failover-Buffer" config="Event-Failover/Default/conf/buffer.xml
" data="Writer[data]" />

<!-- Other element -->
<processing-element name="Element-That-Sends-Data" config="Test/Default/conf/test.xml" data="
Event-Failover-Buffer[data]" />
```

Receiver side:

```
<!-- ACK Writer -->
<processing-element name="Writer" config="Generic-Event-Writer/Default/conf/writer.xml" />

<!-- Event Listener -->
<processing-element name="Listener" config="Generic-Event-Listener/Default/conf/test.xml" data=
"Event-Failover-Responder[data]" />

<!-- Event Failover Responder -->
<processing-element name="Event-Failover-Responder" config="Event-Failover/Default/conf/
responder.xml" command="Writer[data]" data="Element-That-Wait-For-Data[data]" />

<!-- Other element -->
<processing-element name="Element-That-Wait-For-Data" config="Test/Default/conf/test.xml" />
```

2

Event Failover Configuration

The following example shows a standard *Event Failover* buffer configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration ...>
  <session-file>./data/session.eff</session-file>
  <event-buffer transmission-check-period="30" transmission-check-limit="60">
    <disk-spooler fragment-size="10000">
      <fragment-type>array</fragment-type>
      <serializer>
        <class>com.watch4net.events.common.serialization.GenericEventCompactSerializer</class>
      </serializer>
      <deserializer>
        <class>com.watch4net.events.common.serialization.GenericEventCompactDeserializer</
class>
      </deserializer>
    </disk-spooler>
  </event-buffer>
</configuration>
```

- **session-file**: This is the location of the file that will keep the latest acknowledged sequence number. The file must always exist and must contain a valid sequence. On the first launch, use 0-0 as the starting sequence number.
- **event-buffer**: This will specify that this instance of the failover will be a buffer. The *transmission-check-period* will be how often will the buffer check if the responder is still sending acknowledgements and if the acknowledgements are actually increasing. The *transmission-check-limit* will be how much time is needed to consider that not receiving any acknowledgement means that the responder is down or how much time a same acknowledgement can be accepted before detecting that the responder must be forced to continue.
- **disk-spooler**: This is the type of buffer used. In this case, the buffer will be backed-up on disk to allow the buffer to keep its state even when restarted. The other possible value is *memory-spooler*. This buffer type is keeping everything in the runtime memory and should be used only when testing. The *fragment-size* attribute has a default value of **10000** and specifies the size of each fragment of the spooler.
- **fragment-type**: This is the type of fragment that the spooler will use. Currently, the only type available is *array*.
- **serializer**: This is the type of serializer used when backing-up the buffer on disk.
- **deserializer**: This is the type of deserializer used when reading a backed-up file.

The following example shows a standard *Event Failover* responder configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration ...>
  <session-file>./data/session.eff</session-file>
  <event-responder reordering-limit="16000">
    <acking-timerange>1</acking-timerange>
    <acking-eventcount>50</acking-eventcount>
    <!-- preconfigured simple-failure-handler or exception-catcher -->
    <!-- <simple-failure-handler>discarder</simple-failure-handler> -->
    <exception-catcher>
      <exceptions>java.lang.UnsupportedOperationException</exceptions>
      <exceptions>java.lang.IllegalArgumentException</exceptions>
    </exception-catcher>
  </event-responder>
</configuration>
```

- session-file: This is the location of the file that will keep the latest acknowledgement sequence number. The file must always exist and must contain a valid sequence. On the first lunch, use 0-0 as the starting sequence number.
- event-responder: This will specify that this instance of the failover will be a responder. The *reordering-limit* attribute has a default value of **16000** and specifies how many non-ordered events will be kept from the responder side. For example, with a limit of 50, the responder will be able to keep all the values from 0-1 to 0-49 if for any reason the event 0-0 is taking a long time to arrive.
- acking-timerange: This is the time between two sent acknowledgements in seconds.
- acking-eventcount: This is the count of valid events received before sending and acknowledgement.
- exception-catcher: This is the current handler when the responder fails to send the events to the next element. In this case, the responder will still acknowledge the erroneous events if the responsible error is **not** an `UnsupportedOperationException` or an `IllegalArgumentException`. The tag can be replaced by *simple-failure-handler* which must contains one of the failure handlers that does not need configuration. To this date, the only simple failure handler available is *discarder* which will discard erroneous events and acknowledge it.

It is important to find the right balance between the configuration of the buffer and responder. It is best to have a fast acknowledgement configuration and have a reordering limit roughly equal to the multiplication of the size of a fragment of the buffer and the count of senders. If the fragments size are

500 and there's 3 processing elements filling the buffer, the responder should have a reordering buffer size equal to 1500-2000.