



EMC® Watch4net®

Version 6.3u2

JMS Listener Administration 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
2.1 Considerations on Differentiated Data	2
2.2 Workflow Steps	3
2 Event Processing Manager Configuration	5
3 JMS Listener Configuration	7
4 Converters	11
1 BasicTextMessageConverter	11
2 BasicMapMessageConverter	12
3 BasicBytesMessageConverter	12
4 BasicObjectMessageConverter	13
5 Alcatel5620MessageConverter	13
5 Table and Mapping File Configuration	15
1 Create a New JMS Events Table	16
2 Configure a <i>Datastore Writer</i> Mapping File	17
3 Configure a Frontend Mapping File	19
4 Configure the APG.xml file	20

6	Appendix A: XML Schema	23
7	Appendix B: EventSpy	27

Preface

Purpose

The *JMS Listener* is a *Processing Element* designed to subscribe to JMS topics in order to receive messages whenever they are published by a publisher.

Intended Audience

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

Prerequisites

The *JMS 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; version 5.1 or later.

Document Organization

1. Overview	Provides a quick overview of this product, as well as configuration overview steps and considerations.
2. <i>Event Processing Manager</i> Configuration	How to configure the <i>Event Processing Manager</i> to include a <i>JMS Listener</i> .
3. <i>JMS Listener</i> Configuration	How to configure the <i>JMS Listener</i> itself.
4. Converters	Details on the different types of converters and how they create events.
5. Table and Mapping File Configuration	Provides an overview example of configuring tables, matching mapping files, and related configuration steps, necessary when defining a new set of events that will be processed by the <i>JMS Listener</i> .
Appendix A: XML Schema	The XML schema the <i>JMS Listener</i> 's configuration is based on.
Appendix B: EventSpy	Shows how to use an <i>EventSpy</i> to see the field contents of incoming events.

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.

Overview

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

1 Product Overview

With the *JMS Listener* you can use *APG* to listen to JMS servers by subscribing to topics published to their respective queues.

Messages are converted to *APG* events according to the conversion criteria of one of the converters provided, or a custom converter you can create and define.

Output events are then sent within the *APG* event flow to one or more other event processing elements—usually directly to one of *APG*'s *Database Writers*.

2 Configuration Overview

This section includes some information to consider which could affect more complex *JMS Listener* implementations where multiple instances of the *JMS Listener* might be necessary, as well as multiple tables having to be created in the database each with matching configuration files.

If you are planning a simple *JMS Listener* implementation, you can skip to the [Workflow Steps](#) on the next page.

2.1 Considerations on Differentiated Data

The basic implementation of the *JMS Listener* is described in the workflow steps below.

One important factor that could affect your implementation is the consistency of the fields coming from the particular JMS topics that are gathered by the *JMS Listener*.

Most often, when implementing the *JMS Listener* and configuring the third party vendor's JMS server, the data should be normalized and a minimum number of converters in the *JMS Listener* will be able to provide events with a consistent set of fields. This means, for example, that a single *JMS Listener* can be used, and a single table and set of mapping files can be configured, as explained in the workflow below.

Sometimes, even when using multiple converters (see [Converters](#) on page 11), fields are consistent enough to rely on one *JMS Listener* written to one database table via a single *Database Writer*—in this case some events may end up with fields that have a null value since the flow will take into account a superset of fields (as configured) for all events, but again, if null values are kept to a minimum one *JMS Listener* could do the job.

Other times, however, if events have different sets of fields, potentially requiring multiple converters, it makes sense to split the streams, using multiple *JMS Listeners*, and writing to separate database tables using multiple *Database Writers*.

A third option exists if you want to reduce the number of *JMS Listeners* but still write events with different sets of fields to multiple database tables. In this case you can use an *Event Demultiplexer*, which sits between the *JMS Listener* and two or more *Database Writers*, writing to a particular data stream based on which fields exist in the incoming event, with each data stream terminating at a particular *Database Writer* and thereby written to a different table.

When using an *Event Demultiplexer*, you still have to create each unique table, with the corresponding mapping files and *Database Writer* that will be used for each differently composed event, but you can reduce the overall configuration by using only one *JMS Listener*. Contact support@Watch4net.com for more information on the *Event Demultiplexer*.

The options above will affect the workflow configuration explained below.

2.2 Workflow Steps

Follow the workflow steps below to configure the *JMS Listener* and its complement modules:

1. Configure relevant JMS servers to publish topics to their queue. See the appropriate JMS server documentation from the third party vendor.
2. Configure the **processing.xml** file for the Event Manager. See [Event Processing Manager Configuration](#) on page 5.
3. Configure one or more *JMS Listener* configuration files depending on your number of *JMS Listener* instances. Remember that a single *JMS Listener* can listen to many JMS servers, including primary and failover sources, and include multiple converters, so you may only need one instance. See [JMS Listener Configuration](#) on page 7. Also refer to [Converters](#) on page 11 to assist you in your configurations when choosing converters to use.
If you need to process events with quite different fields, potentially using different converters, you would have to create additional corresponding tables in the database with matching mapping files. So you would repeat steps 4 to 8 below, for each table that must be created.

Note:

Steps 4 to 8 may not be necessary if the *JMS Listener* is part of an implementation where database tables and mapping is configured automatically. See the package's documentation for more information. If this is so you can skip to step 9.

4. (Optional) Use an *EventSpy* to find out the field composition of incoming JMS events. See [Appendix B: EventSpy](#) on page 27.
It is recommended to check the third party vendor's documentation first to see what fields are available with the published JMS topics before trying to use an *EventSpy* to gather this information.
5. Create a new table in the events database for JMS events. See [Create a New JMS Events Table](#) on page 16.

6. Configure a new mapping file for the *Datastore Writer* with the required fields. See [Configure a Datastore Writer Mapping File](#) on page 17.
7. Configure a new mapping file for the Frontend with the required fields. See [Configure a Frontend Mapping File](#) on page 19.
8. Configure the **APG.xml** file to point to the right database and Frontend mapping file. See [Configure the APG.xml file](#) on page 20. This step includes adjusting the **server.xml** file if necessary.
9. Configure other elements in the event flow that will receive the event output from the *JMS Listener*.
10. Restart the Event Manager, the Datastore, and any other affected modules or services.

Note:

When creating a custom converter, there will be an extra workflow step which will generally take place before most of these steps. Contact support@Watch4net.com for more information.

2

Event Processing Manager Configuration

To use a *JMS Listener* instance you must modify the *Event Processing Manager's* **processing.xml** file to include a line for each *JMS Listener* 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="DW" config="..." />  
  
<!-- the JMS Event Listener -->  
<processing-element name="JMS"  
config="JMS-Listener/Default/conf/jms-listener.xml" data="DW" />
```

This configuration is applied to the **processing.xml** file, located by default at: <APG>/ Event-Processing/Event-Proc
name>/conf

Like other processing elements, if you use multiple instances of the *JMS Listener* in the event processing flow, they each must be declared in the **processing.xml** for the particular *Event Processing Manager* within which they run. A single *JMS Listener* can process events from many JMS servers.

The *JMS Listener* usually forwards events to a *Database Writer*. Since it exclusively listens to events coming from a JMS server, it accepts no input streams from other processing elements. The following output stream is supported:

- **data (out)**: this is the event output, where the *JMS Listener* sends the events (usually connected to a *Database Writer*'s input data stream).

You can also, for example, indicate here to send events from the *JMS Listener* to multiple *Database Writers* for redundancy by including both of their names in the value of the **data** attribute; or you can send events to other processing elements by similarly entering their name. When entering multiple output destinations, include them all in the quotes separated by a space, for example:

```
<... data="DW1 DW2">
```

3

JMS Listener Configuration

The following example is the default *JMS Listener* configuration. The default *JMS Listener* configuration file is called **jms-listener.xml** and is located at: <APG>/Event-Processing/JMS-Listener/<instance name>/conf. Its configuration is based on the XML schema, located in the same directory and called **jms-listener.xsd**, which is included in this document in Appendix A.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://www.watch4net.com/Events/JMSListener"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/Events/JMSListener_1_0_0/src/conf/com/watch4net/events/processing/listeners/jms/jms-listener.xsd_1_0_0">
  <connection-check-period>30s</connection-check-period>
  <source id="source" client-id="APG-EVENTS@1" >
    <primary connection-factory="JmsTopicConnectionFactory"
      context-factory="org.exolab.jms.jndi.InitialContextFactory">
      <url>url</url>
      <username>username</username>
      <password>password</password>
    <topics>
      <topic>topic</topic>
    </topics>
  </source>
</configuration>
```

```

        </topics>
    </primary>

    <!--
        Defines a list of converters. The order in which the converters are
        defined also defines their priority. Therefore, if two converters
        capable of handling the same type of message are defined,
        the first one that is declared will always handle the messages of the
        given type.
    -->

    <converters>
        <converter class="com.watch4net.events.processing.listeners.jms.converter.
BasicTextMessageConverter" />
        <converter class="com.watch4net.events.processing.listeners.jms.converter.
BasicMapMessageConverter" />
    </converters>
</source>
</configuration>

```

- **connection-check-period** Interval at which the *JMS Listener* will try to reconnect to the primary or failover sources if a failure is detected.
- **source id=id client-id=cliend-id [jms-context=jms-context] [message-filter=message-filter]** A source represents a JMS server instance to monitor. Each source can be composed of a primary source and multiple failover sources, if needed. This is useful for specifying backup JMS servers in case the primary server is down. The *id* parameter is mandatory and can be set to anything, as it is only used for logging purposes. The *client-id* attribute is also mandatory and refers to the unique ID by which the JMS server will be able to identify the connection. The *jms-context* parameter is optional and is only useful if the JMS provider is using a specific JMS context instead of the global context. This information should appear in the provider's documentation. The *message-filter* attribute is also optional, and allows to filter on the incoming messages. The filtering syntax is JMS-specific and is out of the scope of this document, but more information can be found at http://download.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/prog_model.html.

Topics must be configured on the third party vendor's device to be published to its queue, which can then be retrieved by APG.

- `primary/failover` [`context-factory=context`] [`connection-factory=connection`] Determines if the source is the primary source or a failover source.

The *context-factory* attribute is optional, and its default value is *org. exolab. jms. jndi. InitialContextFactory*. The *connection-factory* attribute is also optional, and its default value is *JmsTopicConnectionFactory*. In both cases the default values indicated here are used if not explicitly indicated. These two attributes should only need to be changed if the underlying JMS implementation is changed: Currently, *openJMS* is used. Consult the third party vendor's own documentation for more information on whether different *context-factory* and *connection-factory* values should be used. Any other attribute can also be added in case another JMS implementation requires more attributes.

- * `url` The url of the JMS server's queue (i.e.: `tcp://127.0.0.1:3035`).
- * `username` The username to use to access the JMS server.
- * `password` The corresponding password to use to access the JMS server (leave blank for none).
- * `topics`
 - `topic` A topic to subscribe to. There can be as many `topic` tags as needed.
- `converters`
 - * `converter class=class` Defines a converter that will be used to convert a JMS message into an APG event. Please refer to [Converters](#) on page 11 for a list of available converters.

The `converter` tag can be repeated as many times as necessary to make sure that all message types have a converter. The order in which they are defined also defines their priority: the first converter that is defined will have the highest priority. So for example if two converters capable of handling the same type of message are defined, the first one that is declared will always handle the message of the given type with subsequent ones being ignored.

Multiple Sources The *JMS Listener* can be configured to listen to many sources simultaneously (there is no imposed limit, depending only upon hardware performance). 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 (see example below).

```
...
<source id="Source-1">
  <primary>
    ...
  </primary>
  <failover>
    ...
  </failover>
</source>
<source id="Source-2">
  <primary>
    ...
  </primary>
  <failover>
    ...
  </failover>
  <failover>
    ...
  </failover>
</source>
...
```

Custom converters Since converter implementations are entirely configurable, you could write your own converter to handle specific message types. This can be achieved by implementing the following Java interface: *com.watch4net.events.processing.listeners.jms.converter.MessageConverter*.

The basic converters that are bundled with the listener don't need any configuration. However, in the case when a custom implementation requires to be configured, any XML element can be passed inside the *converter* tag. Upon startup, the converter implementation's *configure* method will be called with the content of this tag as a parameter. It is the implementation's responsibility to validate and read the content of this XML snippet.

For information on how to write a custom converter, you can contact support@Watch4net.com.

Converters

1 **BasicTextMessageConverter**

Supported JMS Message Types

All implementations of `javax.jms.TextMessage`.

Description

Creates an event which contains the text content of the message in the “Message” field.

Java class

`com.watch4net.events.processing.listeners.jms.converter.BasicTextMessageConverter`

Configuration

No configuration required

2 BasicMapMessageConverter

Supported JMS Message Types

All implementations of javax.jms.MapMessage.

Description

Creates an event which contains all the fields in the message map. Each key and associated key value in the message map, becomes a field name with their associated field value in the output event.

Java class

com.watch4net.events.processing.listeners.jms.converter.BasicMapMessageConverter

Configuration

No configuration required

3 BasicBytesMessageConverter

Supported JMS Message Types

All implementations of javax.jms.BytesMessage.

Description

Creates an event which contains the content of the message in the "Message" field as an array of bytes.

Java class

com.watch4net.events.processing.listeners.jms.converter.BasicBytesMessageConverter

Configuration

No configuration required

4 BasicObjectMessageConverter**Supported JMS Message Types**

All implementations of `javax.jms.ObjectMessage`.

Description

Creates an event which contains the content of the message in the “Message” field as a plain java Object.

Java class

`com.watch4net.events.processing.listeners.jms.converter.BasicObjectMessageConverter`

Configuration

No configuration required

5 Alcatel5620MessageConverter

This converter is suitable only when receiving messages from an Alcatel 5620 server. It is tightly coupled with the format used by Alcatel in their JMS implementation. For more information see the *Event Monitoring using JMS* section, in the *Alcatel 5620 OSS Developer Guide*.

Supported JMS Message Types

Alcatel’s implementation of the `ObjectMessage` interface, which consists of a `TextMessage` wrapped in an `ObjectMessage`’s payload.

Description

Creates an event that contains the fields that are configured in the provided configuration.

Java class

com.watch4net.events.processing.listeners.jms.converter.alcatel.Alcatel5620MessageConverter

Configuration

```
<converter-configuration>
  <!-- The XPath expression that points to the event type in the Alcatel message
  --> <type-path>...</type-path>

  <!-- Fields defined here will be used for events of all types -->
  <!-- The name attribute represents the field name in APG -->
  <!-- The value is an XPath expression pointing to the value in the Alcatel
  message -->
  <field name="...">...</field>

  <!-- The name attribute must contain a valid Alcatel message type -->
  <type name="...">
    <!-- Fields defined here will only be used for the specified message type
    -->
    <field name="...">...</field>
    <field name="...">...</field>
  </type>

  <!-- Any number of different Alcatel types can be listed. -->
  <type name="...">
    ...
  </type>
</converter-configuration>
```

5

Table and Mapping File Configuration

This section includes an example configuration overview for additional steps you must perform if you are customizing the *JMS Listener* to listen to particular topics you have defined.

In this case, you will have to create a new SQL table for the JMS events, create related mapping files, and configure the **APG.xml** file correspondingly. You may also have to adjust the **server.xml** file.

This example uses a simple configuration using the the `BasicTextMessageConverter`. See [Converters](#) on page 11.

You have to perform these steps additional times if you have multiple *JMS Listener*'s capturing events with different fields, potentially using different converters.

1 Create a New JMS Events Table

A new table in the *APG* events database must be created for JMS events with the correct fields you want to preserve.

If are using multiple instances of *JMS Listeners* capturing and converting events with different fields, additional corresponding tables must be created.

To create the table in the Datastore, you first need to consult the third party's vendor documentation, then you may need to check the *EventSpy* output, to see which fields are available in the incoming events. See [Appendix B: EventSpy](#) on page 27.

Then create the table by doing the following:

1. On Unix, login to the *Datastore* by using the *datastore-client.sh*.

The script is available in <APG>/bin. The *Datastore* must be installed to proceed with the configuration. It automatically creates this database access script upon installation. For more information, see the *APG-RPE-How-To* PDF.

Use the defaults for username/database (apg, events). The default password is watch4net.

On Windows, enter the following at a command prompt to login to the *Datastore*:

```
C:\Program Files\APG\Databases\APG-Datastore\Default\bin\mysql.exe" --defaults-file="C:\Program Files\APG\Databases\APG-Datastore\Default\my-ib.ini" -u root -p events
```

The default password is watch4net.

2. Enter SQL commands similar to the following, either using *source file_name*, where file_name is the path to the file containing the SQL commands, or directly from the SQL prompt:

```
CREATE DATABASE IF NOT EXISTS events;
GRANT ALL PRIVILEGES ON events.* TO apg@'localhost' IDENTIFIED BY 'watch4net';
GRANT FILE ON *.* TO apg@'localhost' IDENTIFIED BY 'watch4net';

USE events;
DROP TABLE jms_text;
CREATE TABLE IF NOT EXISTS jms_text (
  OpenedAt          INT,
  Source             VARCHAR(64),
  Message            VARCHAR(512)
```

```
) ENGINE=BRIGHTHOUSE;
```

This example creates a table called `jms_text` in the events database, with fields that are recognized by the `BasicTextMessageConverter`. See [Converters](#) on page 11.

The table name will be used subsequently in the configuration steps below.

2 Configure a *Datastore Writer Mapping File*

Create a *Database Writer* XML mapping file to correlate with the events that will be received from the *JMS Listener*.

This file must be located at `<APG>/Event-Processing/APG-Datastore-Writer/<instance name>/conf`. It can be called anything, as long as it is then referenced correctly in the **processing.xml** file. In this example it is called **ds-jms-text.xml**.

You can base the configuration on the example below, which continues to use a simple configuration based on a `BasicTextMessageConverter`.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://www.watch4net.com/Events/APGDatastoreWriter" xmlns:xsi="http://www
.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/Events/APGDatastoreWriter/APGDatastoreWriter.
xsd_">

  <datasource url="jdbc:mysql://localhost:53307/events?autoReconnect=true&
autoReconnectForPools=true"
    username="apg" password="watch4net" testOnBorrow="true" testOnReturn="false"
testWhileIdle="true"
    timeBetweenEvictionRunsMillis="10000" validationQuery="SELECT_1" validationQueryTimeout
="5"
    initConnectionSqls="DELETE FROM_jms_text WHERE_Source_IS_NULL;" />
  <expiration field="ClosedAt" period="1d" timeout="365d" />

  <!-- ***** -->
  <!-- Make sure that the folder specified in location is used ONLY by this writer! -->
  <!-- ***** -->

  <batching location="tmp/jms-text" period="5s">
    <accessor event-timeout="5s" file-timeout="5m" max-file-size="100M" />
  </batching>
</configuration>
```

```
<archiver connection-retry-period="30s" rename-on-corrupt="true" />
</batching>
<mapping table-name="jms_text">
  <!-- The field attribute refers to the field in the Event -->
  <!-- The name attribute refers to the column name in the database. -->
  <!-- If no name attribute is specified the value of the field attribute will be used as
  both the Event field and the column name -->

  <!-- the event key -->
  <key field="OpenedAt" field-type="OCCURRENCE" xsi:type="CustomFieldMapping"
    helper-class="com.watch4net.events.common.processing.database.mapping.
    MillisToSecMapper" />
  <!-- the event properties -->
  <field field="Source" field-type="DEFINITION" data-type="STRING" xsi:type="
    DefaultFieldMapping" />
  <field field="Message" field-type="OCCURRENCE" data-type="STRING" xsi:type="
    DefaultFieldMapping" />

</mapping>

</configuration>
```

In the file above, notice a few things:

- Field-types should be specified (DEFINITION or OCCURRENCE) for each field, and must match the field-type written to the event by the *JMS Listener* at the source or the event will not be written to the database. If nothing is indicated for the field-type it defaults to OCCURRENCE, but this may not be correct so this should be verified. Different converters will define fields as either DEFINITION or OCCURRENCE depending on whether the field is used to uniquely define the event, where each instance of this event will have these DEFINITION fields, or is used to distinguish different instances of the same event, where the OCCURRENCE fields can be different with each instance of the event.
You will be able to determine if a field is a DEFINITION or OCCURRENCE field if Watch4net has prepared a customized JMS solution, for example as part of a ReportPack, if you write a customized converter where field-types must be defined, or by looking at the EventSpy output. See [Appendix B: EventSpy](#) on page 27.
- Each field usually must include a data-type, which can be determined in the same ways as indicated above, for the field-type. In the case where you are using a CustomFieldMapping for an xsi:type, a data-type is not included as you will have to indicate how the field is handled in the helper-class attribute. In this case, it uses a "MillisToSecMapper" which converts milliseconds to

seconds by dividing by 1000. In most cases the "DefaultFieldMapping" is used, handling the field as indicated by the data-type attribute. There are other CustomFieldMapping mappers available.

For more information on field-types and data-types in the *Database Writer's* mapping file, contact: support@Watch4net.com.

3 Configure a Frontend Mapping File

Create a Frontend XML mapping file to correlate with the events that will be received from the *JMS Listener*.

This file must be located at <APG>/Custom/WebApps-Resources/<instance name>/property-mapping. It can be called anything, as long as it is then referenced correctly in the **APG.xml** file. In this example it is called **JMS-TEXT.xml**.

You can base the configuration on the example below, which continues to use a simple configuration based on a BasicTextMessageConverter.

Property mapping files identify event field names in the database and correlate them with a Frontend property name equivalent. So, in this example, you will find the following properties, as indicated for the display name tags available for the corresponding JMS events in the Frontend: OpenedAt, OpenedAtKey, Source, Message.

```
<?xml version="1.0" encoding="UTF-8"?>
<properties key-separator="-">

  <property name="com.watch4net.apg.v2.modules.database.GenericFlow.properties.
FormattedTimestamp" timestamp="true">
    <name class="string">OpenedAt</name>
    <table class="string">\${table}</table>
    <display-name class="string">OpenedAt</display-name>
  </property>

  <property name="com.watch4net.apg.v2.modules.database.GenericFlow.properties.
GraphableValue" default="true" key="true">
    <name class="string">OpenedAt</name>
    <table class="string">\${table}</table>
    <display-name class="string">OpenedAtKey</display-name>
  </property>
```

```

    <property name="com.watch4net.apg.v2.modules.database.GenericFlow.properties.
StringValue">
        <name class="string">Source</name>
        <table class="string">\${table}</table>
        <display-name class="string">Source</display-name>
    </property>

    <property name="com.watch4net.apg.v2.modules.database.GenericFlow.properties.
StringValue">
        <name class="string">Message</name>
        <table class="string">\${table}</table>
        <display-name class="string">Message</display-name>
    </property>
</properties>

```

In the above example the database property OpenedAt was mapped twice. If you look at the property names you can see why: the timestamp is used from the first property but is rendered as a "FormattedTimestamp". "FormattedTimestamp" properties use characters such as spaces that are not allowed in the key. The key can only have alphanumeric characters, or "-" or "." characters.

Therefore in this case, the OpenedAt property is mapped a second time, and in this instance used as the as both the "default" graphable property, plotted in graphs for this event if no other property is indicated in the Frontend, as well as the event's key. For more information on creating and editing property mapping files for the Frontend, see the *APG-Event-Mapping* PDF.

4 Configure the APG.xml file

You must configure the **APG.xml** file to recognize any new property mapping you created.

The **APG.xml** file is located at:

```
<APG>/Web-Servers/Tomcat/<instance name>/conf/Catalina/localhost
```

Following along with the example in this section, the following sample code can be the basis for your own **APG.xml** configuration:

```

<!-- Gives access to jms_text. -->
<ResourceLink name="jdbc/FLOW-JMS-TEXT:{_mapping=JMS-TEXT.xml,_table=jms_text_}"
    global="jdbc/FLOW-RPE2" type="javax.sql.DataSource" />

```

Take note of the following:

- The configuration above references a FLOW-RPE2 database which must be enabled in the **server.xml** file by commenting the following connection from there:

```
<Resource name="jdbc/FLOW-RPE2" auth="Container" type="javax.sql.DataSource" maxActive="30" maxIdle="5"
    validationQuery="SELECT_1" testOnBorrow="false" testWhileIdle="true"
timeBetweenEvictionRunsMillis="10000" validationQueryTimeout="5"
    minEvictableIdleTimeMillis="60000" maxWait="10000" username="apg" password="
watch4net" driverClassName="com.mysql.jdbc.Driver"
    removeAbandoned="true" removeAbandonedTimeout="60" logAbandoned="true"
url="jdbc:mysql://localhost:53307/events?autoReconnect=true&tinyInt1isBit=
false" />
```

- If by any chance you configure a differently named resource name in a connection instance in the **server.xml** file, use that same name for the `global` attribute in the **APG.xml** file.
The path to the **server.xml** file is: <APG>/Web-Servers/Tomcat/<instance name>/conf

6

Appendix A: XML Schema

The following is the XML schema code the *JMS Listener* configuration file is based on.

The schema is in the file **jms-listener.xsd** located at: <APG>/Event-Processing/JMS-Listner/<instance name>/conf.

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

  <xs:complexType name="Configuration">
    <xs:sequence>
      <xs:element name="connection-check-period" type="duration" />
      <xs:element name="source" type="Source" minOccurs="1"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!--
```

A source is a combination of a primary JMS server and a list of optional failover servers. If the primary server fails to respond, the listener will try to connect to the failover servers in the order in which they are defined.

```

-->
<xs:complexType name="Source">
  <xs:sequence>
    <xs:element name="primary" type="SourceDefinition"
      minOccurs="1" maxOccurs="1" />
    <xs:element name="failover" type="SourceDefinition"
      minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="converters" type="ConverterList" />
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="client-id" type="xs:string" use="required" />
  <xs:attribute name="message-filter" type="xs:string" use="optional" />
  <xs:attribute name="jms-context" type="xs:string" use="optional" />
</xs:complexType>

<xs:complexType name="SourceDefinition">
  <xs:sequence>
    <xs:element name="url" type="xs:string" />
    <xs:element name="username" type="xs:string" />
    <xs:element name="password" type="xs:string" />
    <xs:element name="topics" type="TopicList" />
  </xs:sequence>
  <xs:attribute name="context-factory" type="xs:string"
    use="optional" default="org.exolab.jms.jndi.InitialContextFactory" />
  <xs:attribute name="connection-factory" type="xs:string"
    use="optional" default="JmsTopicConnectionFactory" />
  <xs:anyAttribute processContents="lax" />
</xs:complexType>

<xs:complexType name="TopicList">
  <xs:sequence>
    <xs:element name="topic" type="xs:string" minOccurs="1"
      maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<!--
  Defines a list of converters. The order in which the converters are

```

defined also defines their priority. Therefore, if two converters capable of handling the same type of message are defined, the first one that is declared will always handle the messages of the given type.

```
-->
<xs:complexType name="ConverterList">
  <xs:sequence>
    <xs:element name="converter" type="Converter" minOccurs="1"
      maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Converter">
  <xs:sequence>
    <xs:any processContents="skip" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="class" type="xs:string" use="required" />
</xs:complexType>

<xs:simpleType name="duration">
  <xs:restriction base="xs:string">
    <xs:pattern value="([0-9]+(d|h|m|s|ms))+|([0-9]+)" />
  </xs:restriction>
</xs:simpleType>

<xs:element name="configuration" type="Configuration" />
</xs:schema>
```


Appendix B: EventSpy

With the *EventSpy* utility, you can see all the field names, their values and types, for incoming events.

It is recommended to first consult the third party vendor's documentation to see what event fields are available from that particular source before trying the *EventSpy*.

The *EventSpy* will not always be able to return the fields of the incoming event, depending on their composition, and the converter used.

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 from the *JMS Listener* to the *EventSpy* input:

```
<processing-element name="SPY" config="Event-Processing-Utills/Default/conf/" type="
EventSpy" /> -->
<processing-element name="JMS" config="JMS-Listener/Default/conf/jms-listener.xml" data="
SPY" />
```

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 *JMS Listener*, indicating fields and types when creating tables, and configuring mapping files.