



EMC® Watch4net®

Version 6.3u2

JMX Collector Admin Guide

P/N: pending

Rev : 40030

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
Common Abbreviations and Acronyms	iv
Where to get help	v
1 Overview	1
1 Features	1
1.1 Get attributes from MBeans	1
1.2 Get operation return from MBeans	1
1.3 Custom objects	2
1.4 Transformations	2
1.5 Simple authentication	2
1.6 Collect from multiple sources	2
2 Configuration	3
1 JMX Configuration	3
1.1 Default authentication	5
1.2 Default polling	5
1.3 Locations	5
2 JMX Data Configuration	6
2.1 MBean	8
2.2 Bean	8
2.3 Collection	8
2.4 Properties	9
2.5 Values	9
2.6 Attributes	9
2.7 Methods	10

- 2.8 Constants 11
 - 2.9 Customs 11
 - 2.10 Mapped 11
 - 2.11 Transformations 12
 - 2.12 Relations 12
- 3 Example 13**
 - 1 Basic information 13
 - 2 Configuration 14
 - 2.1 Base configuration file 14
 - 2.2 Data configuration file 14
 - 2.3 Result 15

Preface

Purpose

The purpose of this guide is to provide information about the *JMX Collector* and the way to use it. This guide includes all necessary information to configure and run the collector.

Intended Audience

This guide is intended to be read by any administrator who will need to collect raw values from a JMX enabled JAVA application. Strong JAVA and JMX knowledge is required.

Prerequisites

Watch4net's *APG* Backend and Frontend, version 5.0 or up, must be installed.

Document Organization

1. Overview	This section describes the features of the <i>JMX Collector</i>
2. Configuration	This section covers the configuration of the <i>JMX Collector</i>
3. Example	This section provides an example of the <i>JMX Collector</i> in action

Common Abbreviations and Acronyms

<i>JMX</i>	Java Management Extension
------------	---------------------------

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 *JMX Collector* has been designed to collect raw values from JMX enabled applications.

1 Features

1.1 Get attributes from MBeans

The *JMX Collector* can easily generate raw values and properties from MBean attribute values.

1.2 Get operation return from MBeans

Sometimes attributes may not give all the necessary data, so operations are used. The *JMX Collector* can also generate raw values and properties from the returned value of a called method. Any method with any number of parameters can be called. Parameter types are limited to a defined list; see section [2.7](#).

1.3 Custom objects

Objects returned from MBean attributes and methods can be of any type, including custom types. The only condition is the object must be in the classpath of the *JMX Collector*.

1.4 Transformations

The *JMX Collector* can transform values obtained from attributes and operations using regular expressions. Currently, arithmetic transformations are not supported. Multiple transformations can be applied on the same value.

1.5 Simple authentication

The *JMX Collector* uses simple authentication provided by JAVA JMX.

1.6 Collect from multiple sources

Data can be collected from multiple applications on separate servers using one collector. This allows for the reuse of the same configuration files for a similar collect.

2

Configuration

There are two types of files to configure for the JMX Collector. The *jmx-collector.xml* file contains basic values, and one or more *jmx-collector-data.xml* files contain data-specific values.

1 JMX Configuration

The root configuration file for the *JMX Collector* is named *jmx-collector.xml*. To use the *JMX Collector*, the APG-Collector-Manager configuration file *collecting.xml* must refer to this file. Here is an example:

```
<!-- This is the JMX-collector -->
<collector name="JMX-collector" next="Backend" config="JMX-Collector/Default/conf/jmx-collector.xml" />
```

The JMX configuration file contains credentials, polling options for the collects, and links to data configuration files. It is located at: *APG/Collecting/JMX-Collector/Default/conf/jmx-collector.xml*

Note: All JMX-Collector configuration files are located in its `conf` directory.

Here is an example of *jmx-collector.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<jmx-collector-configuration
  xmlns="http://www.watch4net.com/APG/Collector/JMXCollector"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Collector/JMXCollector_jmx-collector.xsd"
>

  <default-authentication>
    <login>apg</login>
    <password>changeme</password>
  </default-authentication>

  <default-polling>
    <group>group</group>
    <period>240</period>
    <connection pool-size="5"/>
    <properties refresh="5"/> <!-- refresh properties every 5 polling -->
  </default-polling>

  <locations>

    <location id="server-to-poll">
      <jmx-url>service:jmx:rmi:///jndi/rmi://localhost:9998/jmxrmi</jmx-url>
      <specific-authentication>
        <login>apg</login>
        <password>password</password>
      </specific-authentication>
      <specific-polling>
        <properties refresh="10"/>
      </specific-polling>
      <data-configuration-file>./conf/example-jmx-data.xml</data-configuration-file>
    </location>

  </locations>
</jmx-collector-configuration>
```

1.1 Default authentication

This tag defines the default login and password to connect to each location.

1.2 Default polling

This tag defines the default polling options for each location.

group Aggregation and retention group.

period Polling period in seconds.

connection pool size Maximum concurrent threads running.

properties refresh When to refresh properties. Two notation formats can be used. The first type: a positive integer (n) to call refresh properties on (n) polls. For example: (5) means refresh properties each (5) polling intervals. The second type: a time and a delay formatted like this: HH:MM/(seconds). For example: 00:00/21600 means that properties will be refreshed each midnight (00:00) and every six hours (21600 seconds is equal to 6 hours).

Note: When refreshing properties on a scheduled time using the second format type explained above, the refresh actually only occurs at the next polling interval. For example, if the polling period is 720 (15 minutes) and the properties refresh option is set to 00:00/21600, the first refresh of each day will occur between 00:00 and 00:15 at the same time as the polling.

1.3 Locations

Each location tag represents a different JMX enabled JAVA application.

jmx url The JMX URL to connect to the JAVA Virtual Machine exposing MBeans. It is required.

specific authentication If default authentication does not apply for this location, specific credentials can be defined. If the default is correct, no need to use this tag at all.

specific polling Used to override default polling options.

data configuration file Path to the data configuration file. More than one tag can be used to split the configuration into multiple files. At least one is required.

2 JMX Data Configuration

The data configuration file contains all information to create values and properties from exposed attributes and operations.

Here is an example of *jmx-collector-data.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<jmx-collector-data-configuration
  xmlns="http://www.watch4net.com/APG/Collector/JMXCollectorData"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Collector/JMXCollectorData_jmx-collector-
data.xsd">

  <mbean query="myDomain:type=myType">
    <properties>
      <property name="propertyName1">
        <attribute name="attributeName1"/>
      </property>
      <property name="propertyName2">
        <attribute name="attributeName2"/>
      </property>
    </properties>
    <values>
      <value name="valueName1" separator="-" unit="unitOfValue1" variable="propertyName1_
propertyName2">
        <method name="methodName">
          <param type="String" value="stringValue"/>
          <param type="int" value="1"/>
        </method>
      </value>
      <value name="valueName2" variable="propertyName2">
        <method name="methodNameWithNoParams" />
      </value>
    </values>
    <relations>
      <relation id="aBean">
        <attribute name="aBeanAttribute"/>
      </relation>
    </relations>
  </mbean>

  <mbean query="myDomain:type=myOtherType,name=myName">
```

```

<properties>
  <property name="aProperty">
    <method name="methodName"/>
    <transformation match="RegexToMatch" replace="ReplaceWith"/>
  </property>
  <property name="objectName">
    <constant name="ObjectName"/>
  </property>
</properties>
<values>
  <value name="aValue" separator="-" unit="someUnit" variable="aProperty">
    <method name="getAValue" />
    <transformation match="RegexToMatch1" replace="ReplaceWith1"/>
    <transformation match="RegexToMatch2" replace="ReplaceWith2"/>
  </value>
</values>
<relations>
  <relation id="aCollection">
    <attribute name="aCollectionAttribute"/>
  </relation>
</relations>
</mbean>

<bean id="aBean" keep-properties="true">
  <properties>
    <property name="otherProperty">
      <attribute name="prop"/>
    </property>
  </properties>
  <values>
    <value variable="propertyName2_otherProperty" name="aBeanValue">
      <attribute name="val"/>
    </value>
  </values>
</bean>

<collection id="aCollection">
  <properties>
    <property name="colProperty">
      <attribute name="propCollection"/>
    </property>
  </properties>

```

```
<values>
  <value variable="colProperty" name="colValue">
    <attribute name="colValue"/>
  </value>
</values>
<relations>
  <relation id="aBean">
    <attribute name="aBean"/>
  </relation>
</relations>
</collection>

</jmx-collector-data-configuration>
```

2.1 MBean

MBeans are the entry points to collect data. The attribute and method tags inside it refer to exposed attributes and operations from the specified MBean. These are the tags most commonly used.

2.2 Bean

Beans are single JAVA objects of any type. This tag is used for advanced configuration. They are mostly used when returned objects from MBeans are not standard JMX objects. All objects returned must be in the classpath of the *JMX Collector*. Place the required .jar libraries in the ./custom-lib/ directory of the *JMX Collector*.

2.3 Collection

Collections are JAVA objects implementing java.util.Collection or Java arrays. This tag is used for advanced configuration. A Collection tag will extract properties, values and relations from each element contained in the collection object. They are mostly used when returned objects from MBeans are not standard JMX objects. All objects returned must be in the classpath of the *JMX Collector*. Place the required .jar libraries in the ./custom-lib/ directory of the *JMX Collector*.

2.4 Properties

These are the properties binded with the value. The name given is the name of the property as it will be seen in the frontend. Properties are created from attributes or methods.

Note: If Property name exceeds the maximum length, it will be truncated by the APG-Backend (8 by default, see your APG-Backend configuration for the exact value).

2.5 Values

Values are created from attributes and methods. One raw value will be created for each value. Values should be of `float` type. If the attribute or method returns another type, the *JMX Collector* will try to convert it to `float` from the `String` representation of the object.

Technically this is how non float objects will be converted to float:

```
Object o = getAttribute("myAttribute");  
float value = Float.valueOf(o.toString());
```

If the value can't be converted to `float`, an error will be logged and the processing for this connection will end without finishing polling other values and properties.

The `name` attribute is a value specific property used to create the id of the raw value. The `variable` attribute is a space separated list of properties used to create the raw value with the `name`. `name` and `variable` attributes are mandatory. The `unit` attribute is a value specific property to set the unit of the value. It is optional. The default of the `separator` attribute is `"-"`. It is used as a separator when creating the raw value id.

2.6 Attributes

Attributes refer to Java object attributes. In the case of MBeans, attributes must be readable. For the other cases (Beans and Collections) they are like standard Java attributes. Attributes are case sensitive. If the attribute cannot be found, an error will be logged and the current polling to this specific Java Virtual Machine will stop. When the choice occurs, attributes should always be privileged against methods.

2.7 Methods

Methods refer to Java object methods. Methods with parameters are also supported with a limited list of parameter types:

- `String` `java.lang.String` type
- `int` primitive type `int`, does not allow null value
- `long` primitive type `long`, does not allow null value
- `float` primitive type `float`, does not allow null value
- `short` primitive type `short`, does not allow null value
- `byte` primitive type `byte`, does not allow null value
- `boolean` primitive type `boolean`, does not allow null value, possible values are "true" and "false" only
- `double` primitive type `double`, does not allow null value
- `char` primitive type `char`, does not allow null value
- `Integer` `java.lang.Integer` type
- `Long` `java.lang.Long` type
- `Float` `java.lang.Float` type
- `Short` `java.lang.Short` type
- `Byte` `java.lang.Byte` type
- `Boolean` `java.lang.Boolean` type
- `Double` `java.lang.Double` type
- `Character` `java.lang.Character` type
- `Enum` Any enumeration value with complete class path. For example, an enumeration `com.example.SomeEnum` with value `VALUE1` will be written `com.example.SomeEnum.VALUE1`.

Note: To use Enum, the enumeration must be present in the classpath of the *JMX Collector*. To do so, place the required `.jar` libraries in the `custom-lib` directory of the *JMX Collector*.

All other possible types are not currently supported. If the `value` attribute of a `param` tag is missing, the parameter value will be set to null.

WARNING: Primitive types do not allow `null` value. Using this value will log an error and stop the current polling

for the selected connection.

2.8 Constants

Constants are special tags which apply only to MBean sections. They may be useful as properties (and only for properties). The following lists the defined constants:

- `JMX-URL` This will return the JMX URL as entered in the configuration.
- `LocationID` This will return the location ID as entered in the configuration for each specific location.
- `Description` This will return the description information obtained from the specific MBean.
- `ObjectName` This will return the objectname information obtained from the specific MBean.
- `ClassName` This will return the classname information obtained from the specific MBean.
- `Source` This will return the collector name for this specific instance.

2.9 Customs

Customs are custom tags used to create any property giving a desired value. For example, if you want a property named "part" with its value "Java Application", it will look like this:

```
...  
    <property name="part">  
        <custom value="Java Application"/>  
    </property>  
...
```

It can also be used to create a value with chosen value to handle some specific cases. For example, if an MBean offers interesting properties but no values, it is possible to create a custom value with value NaN (case sensitive) to tell the backend to store the properties only.

2.10 Mapped

Mapped tags are used to import all properties contained in a map where the property name is the map's entry key and the property value the map's entry value. The name attribute of the property and mapped elements can be arbitrary values as they are not used to fetch the map. The Mapped element

only accepts attribute and method tags to fetch the map. If maps do not contain strings as keys and values, the objects found will be converted to strings.

Here is an example to fetch properties in a map from attribute Props.

```
...  
    <property name="PropsFromMap">  
        <mapped name="PropsFromMap">  
            <attribute name="Props"/>  
        </mapped>  
    </property>  
...
```

2.11 Transformations

When attributes or methods do not return data in the desired format, the transformations feature may be useful. Transformations allow regular expression find and replaces. If the expression cannot be found, the transformation will not be applied. Multiple transformations can be applied in the same order that they are defined. For example, using this transformation:

```
<transformation match="([0-9]+)ms" replace="$1"/>
```

This transformation will remove `ms` after the digits. Input `4564ms` will output as `4564`.

2.12 Relations

Relations are the links between objects to poll. The next object can be called with attribute or method the same way as for values and properties. The `id` must match with only one `Bean` or `Collection` tag for all data configuration files depending on the same base configuration file.

3

Example

This is an example to poll uptime from a Java application exposing MBeans with JMX.

1 Basic information

The application we poll is located at this (fictive) JMX URL:

`service:jmx:rmi:///jndi/rmi://ApplicationServer:9997/jmxrmi.`

The uptime will be the value and `startTime` and `VmName` will be properties. The JMX query to get the right MBean is `java.lang:type=Runtime`.

Login and password are `mylogin` and `mypassword`.

Polling will take place every 5 minutes with properties refreshed every 5 polling cycles.

2 Configuration

According to the above information, the configuration is as follows.

2.1 Base configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<jmx-collector-configuration
  xmlns="http://www.watch4net.com/APG/Collector/JMXCollector"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Collector/JMXCollector_jmx-collector.xsd"
>

  <default-authentication>
    <login>mylogin</login>
    <password>mypassword</password>
  </default-authentication>

  <default-polling>
    <group>group</group>
    <period>240</period>
    <connection pool-size="5"/>
    <properties refresh="5"/> <!-- refresh properties every 5 polling -->
  </default-polling>

  <locations>

    <location id="server-to-poll">
      <jmx-url>service:jmx:rmi:///jndi/rmi://ApplicationServer:9997/jmxrmi</jmx-url>
      <data-configuration-file>./conf/example-jmx-data.xml</data-configuration-file>
    </location>

  </locations>
</jmx-collector-configuration>
```

2.2 Data configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<jmx-collector-data-configuration>
```

```
xmlns="http://www.watch4net.com/APG/Collector/JMXCollectorData"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.watch4net.com/APG/Collector/JMXCollectorData_jmx-collector-
data.xsd">

  <mbean query="java.lang:type=Runtime">
    <properties>
      <property name="starttime">
        <attribute name="StartTime"/>
      </property>
      <property name="vmname">
        <attribute name="VmName"/>
      </property>
    </properties>
    <values>
      <value name="uptime" variable="vmname">
        <attribute name="Uptime" />
      </value>
    </values>
  </mbean>

</jmx-collector-data-configuration>
```

2.3 Result

Every 5 minutes the *JMX Collector* will connect to the JMX URL and ask for the MBean specified by the query. It will get the value (uptime) and the two properties (startTime and VmName) and send them as raw values to the APG-Backend. Every 5th polling cycle, the raw value will carry the refresh flag to refresh the properties.

Note: *jconsole* is a tool included with the Java JDK package that can connect to a Java application and show all exposed MBeans. This tool is very useful to find queries for `mbean` tags. It lists also attributes, methods and descriptions about them. For more details about *jconsole* see the SUN documentation online at:
<http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html>