WebSphere Message Broker V7

SupportPac IAM8
Java Management Extensions Node

Marc Fiammante, DE, Europe SWG Solutions Lead Architect
Jochen Benke, ISSW WebSphere BI Services

Take Note!

Before using this report, be sure to read the general information under "Notices".

First Edition, November 2011

This edition applies to WebSphere Message Broker V7.0 – Java Management Extensions Node V1.0 and to all subsequent releases and modifications unless otherwise indicated in new editions.

Notices

The following paragraph does not apply in any country where such provisions are
inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain
transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that
IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not
intended to state or imply that only IBM's program or other product may be used. Any
functionally equivalent program that does not infringe any of the intellectual property
rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those
expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this
document. The furnishing of this document does not give you any license to these
patents. You can send license inquiries, in writing, to the IBM Director of Licensing,
IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test
and is distributed AS-IS. The use of the information or the implementation of any of

these techniques is a customer responsibility and depends on the customer's ability to
evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- WebSphere MQ (WMQ)
- WebSphere Message Broker (WMB)
- WebSphere Application Server (WAS)

The following terms are trademarks of other companies:

- Windows 2000 Microsoft Corporation

Document Change History

Document Number Date	Date	Description of Changes
1.0.0	11/01/11	Initial Release

Introduction

Service Oriented Architectures are heterogenous landscapes of software components. Many of these building blocks based on J2EE and these have normally a remote Java Management Extensions (JMX). A Interface to controll J2EE servers, applications or cells. The JMX interface is realized in a three tier architecture.

The probe tier contains the Managed Beans (MBeans) which provides methods and attributes to control the components.

The Agent level (MBeanServer) is the engine of JMX. It is an intermediary between MBeans and connecting applications.

The Remote Management level is interface to the JMX client. The client can use connectors and adapters for accessing the applications. This can be done remote and local while the remote connection is done by using JSR160.

The JMX adapter node is message broker partial implementation of the JSR160. The node queries JMX objects from the MBeanServer and it runs methods or returns attributes as result of the query. The input is described in a XSD schema file from which a messages set is generated to control the input tree. The JMX Adapter Node add the result to the input structure and this can be used for further processing in the flow. The node doesn't run in the transaction of the flow. For example start or stop operation are not roll back in case of a flow exception appears after passing successfully the JMX Node.

Possible uses

The JMX Adapter node can be used for

- call MBean functions
- query MBean fields

The XML representation of the input and output tree is equivalent to the delivered JMX Adapter Node XSD. The XSD is describe later.

Overall characteristics of the JMX Adapter node

The JMX Adapter node is implemented in Java, as a WebSphere Message Broker User Defined Node. Following installation of this SupportPac in the WebSphere Message Broker Toolkit, the node is added the WebSphere Message Broker toolkit palette as JMX Request element . This node can be included in flows and sub flows, and deployed as part of them.

The node hasn't any properties because all needed setting are part of the input tree. This input tree must be match the delivered JmxCall.xsd.

The connections to application servers are cached in the Java Virtual Machine (JVM) memory of the execution group. The key is user id, password, jmxUri and VM arguments for a specific connection in the cache.

Other Characteristics

The JMX Request Node does not delete any elements in the input tree. The output tree matches the input tree extended by queried result of functions and attributes. The results will placed relative to the according request. You will have one result per request in multi JMX call structures. Request can change the behavior of the Application Server this isn't done in transaction of the flow. This means exceptions which raise in flows afterward don't roll back the application server.

Limitations

The current version of the node doesn't support JMX function arguments which are specific to the application server implementation. For example com.ibm.websphere.management.Session is such not supported Object which is used to change the application server structure.

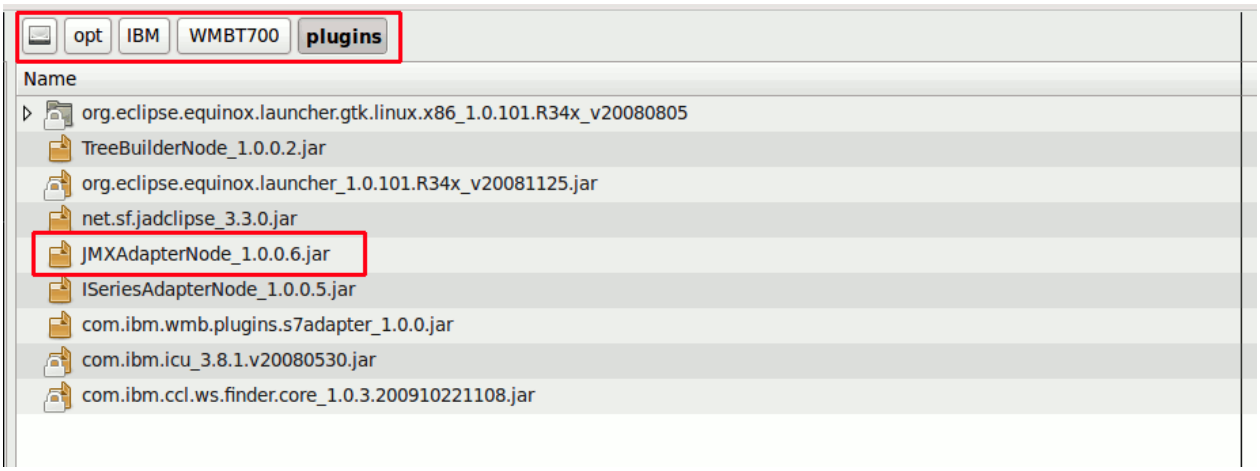
Prerequisites

This SupportPac requires WebSphere Message Broker Toolkit version 7.0.

Installation

Installing the SupportPac in the Toolkit

This SupportPac for the JMX Adapter Node is available as a .zip file, which can be downloaded from the IBM web site for "WebSphere MQ - SupportPacs by Product". The SupportPac Number is IAM8.



- Unzip the .zip file to a temporary directory, and copy the JMXAdapterNode_*.jar file that it contains to the Plugins directory that is part of your WebSphere Message Broker Toolkit installation. By default, this will be the directory: <Toolkit_RootInstallDirectory>/plugins on the Windows or Linux workstation where the WebSphere Message Broker Toolkit is installed. This .jar file contains the JMX Request User Defined Node.
- Start the Toolkit.
- The JMX Request Node should now be available for use as part of the Toolkit Node Palette.

Uninstalling the Support Pac in the Toolkit

- Stop the WebSphere Message Broker Toolkit.
- Locate the JMX Request jar file in the Toolkit Plugins directory.
- Delete the JMX Request jar file.
- Restart the WebSphere Message Broker Toolkit.

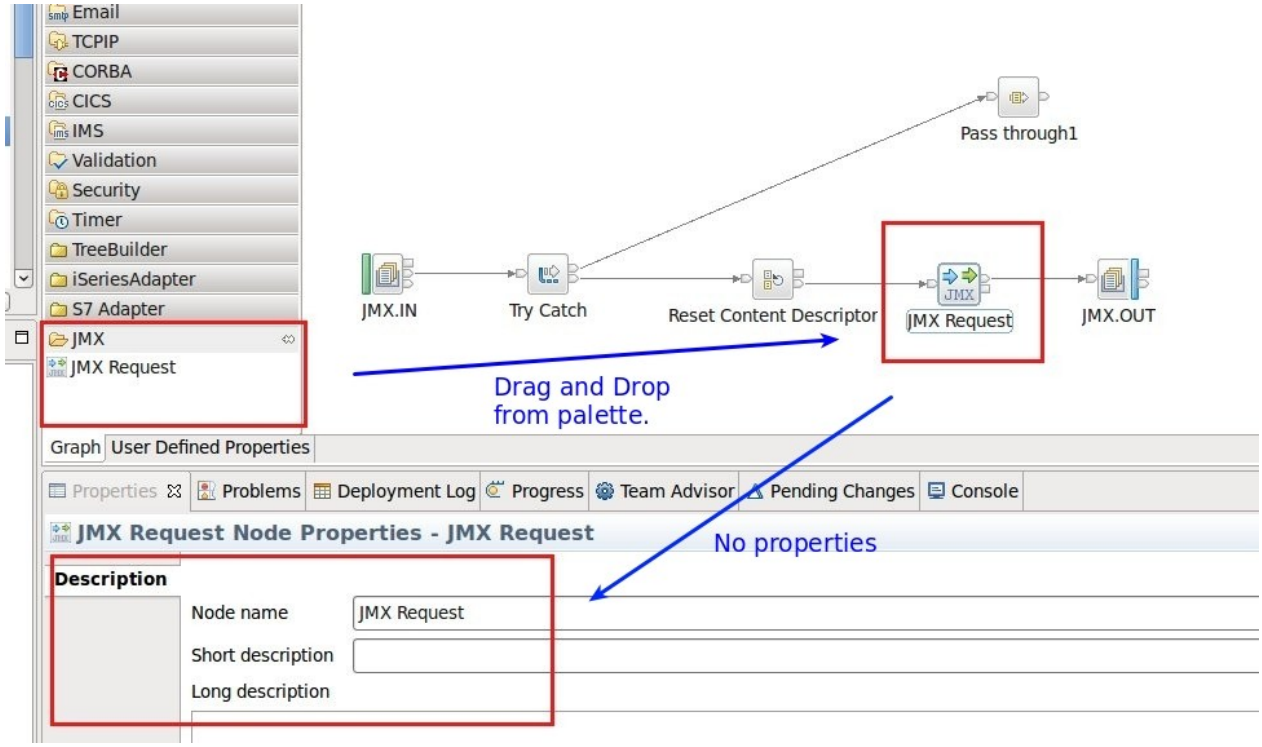
Installing the node at the broker

- Unzip the .zip file to a temporary directory, and copy the JMXAdapterImpl.jar file that it contains to the lil path of your WebSphere Message Broker.
- Copy the JSR160 implementation jars of the application server to the shared-classes directory of your WebSphere Message Broker.

User interface for developers

Once the installation of the SupportPac as part of the WebSphere Message Broker Toolkit is complete on the developer's workstation, the developer will be able to drag and drop the JMX Request Node from the Node Palette, and use it just like all the other WebSphere Message broker Toolkit nodes.

The user interface is illustrated in the following diagram.



Developer's Guide

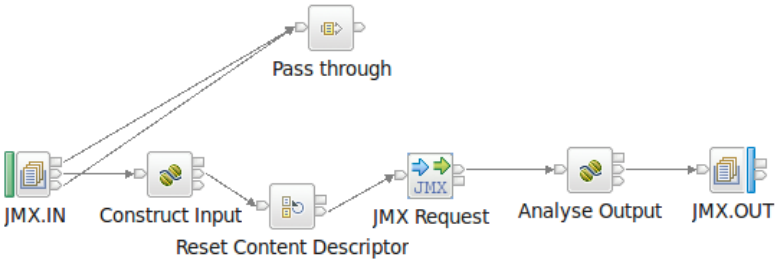
The following paragraphs illustrate, step-by-step, how to use the JMX Request node.

Application Server Security disabled

Add the JMX Request node from the palette as part of a Message Broker flow. This will generally be in a flow after using some other a transformation node, as illustrated below. There are no Node Properties to specify for the JMX Request Node. A Reset Content Descriptor Node controls the input tree after building the request message. Afterwards the JMX request result is analyzed in another transformation node. The example is with turned off application server security.

For WAS 7.0.0.7 make the Administration Thin Client JAR file available by copying com.ibm.ws.admin.client_7.0.0.jar from a WebSphere® Application Server environment to the shared-classes folder of the broker.

The following diagrams, showing the message flow. The scenario is stopping a specific Message Driven Bean Listener Port and getting the maxMessages Value of that property.



The construction of a JMX request (Construct Input) is done in Java like described in the following listing. The grammar of the input structure is explained in detail in chapter *JMX Adapter Node Message Format.*

.

```
// get the message root and set the parser
MbElement mbElement = outMessage.getRootElement();
mbElement = mbElement.createElementAsLastChild(MbXMLNSC.PARSER_NAME);

// create one jmx call request
mbElement = mbElement.createElementAsLastChild(MbElement.TYPE_NAME, "jmxCalls", null);
mbElement = mbElement.createElementAsLastChild(MbElement.TYPE_NAME, "jmxCall", null);

// set the credentials for the call
MbElement credentialElement = mbElement.createElementAsLastChild(MbElement.TYPE_NAME, "credentials", null);
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "userId", "admin");
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "jmxUri", "service:jmx:rmi://localhost:2811/jndi/JMXConnector");
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "providerUri", "corbaloc:iiop://localhost:2811");
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "initialContextFactory", "com.ibm.websphere.naming.WsnInitialContextFactory");

// specify the jmx query
MbElement queryStringElement = mbElement.createElementAsLastChild(MbElement.TYPE_NAME, "queryString", null);
queryStringElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "value", "WebSphere:type=ListenerPort,name=MY_TEST,*");

// specify the operation to call
MbElement operationElement = queryStringElement.createElementAsLastChild(MbElement.TYPE_NAME, "operation", null);
operationElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "value", "stop");
operationElement.createElementAsLastChild(MbElement.TYPE_NAME, "arguments", null);

// specify the attribute to query
MbElement attributesElement = queryStringElement.createElementAsLastChild(MbElement.TYPE_NAME, "attributes", null);
MbElement attributeElement = attributesElement.createElementAsLastChild(MbElement.TYPE_NAME, "attribute", null);
attributeElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "name", "maxMessages");
```

After calling the JMX request node you get the input structure back extend by a new tag. This tag has the name result and is a child of jmxCall. To analyze the input you have to add the delivered JMXResult.jar to the Java Project of the flow. Here you find a static function for decoding the contents of the result tag to a java.util.Vector. The vector contains JMXResult objects.

```
// query the result tag of the call
MbXPath resultXPath = new MbXPath("/jmxCalls/jmxCall/result/serializedJavaObject");

List<MbElement> nodeset = (List<MbElement>) inMessage.evaluateXPath(resultXPath);

for (Iterator<MbElement> iterator = nodeset.iterator(); iterator.hasNext();) {
    try {
        // use the function to transform the base64 string to a Vector
        Vector<JMXResult> returnVector = JMXHelper.getResultVector((String)((MbElement) iterator.next()).getValue());

        for (Iterator<JMXResult> iterator2 = returnVector.iterator(); iterator2.hasNext();) {
            JMXResult jmxResult = (JMXResult) iterator2.next();

            // get the values from the attribute query
            String name = jmxResult.getObjectName().getCanonicalName();
            Integer propertyValue = (Integer) jmxResult.getAttributes().get("maxMessages");

            MbElement childElement = mbElement.createElementAsLastChild(MbElement.TYPE_NAME, "ListenerPort", null);

            childElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "name", name);

            childElement = childElement.createElementAsLastChild(MbElement.TYPE_NAME, "Property", null);
            childElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "name", "maxMessages");
            childElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "value", propertyValue);
        `
```

Application Server Security enabled

You have to configure the following steps for accessing a application server with enabled security. This example shows this for WebSphere Application Server 7.0.0.7. You have to configure additional steps for the broker user environment and extend the JMX Request from the first example. For other application server types you have to refer the specific server documentation. Proceed the following steps and restart the broker.

- Make the Administration Thin Client JAR file available by copying com.ibm.ws.admin.client_7.0.0.jar from a WebSphere® Application Server environment to the shared-classes folder of the broker.
- Make the security JAR file available by copying com.ibm.ws.security.crypto.jar from a WebSphere® Application Server environment to the shared-classes folder of the broker.
- To use JSR160RMI connectors, copy the sas.client.props file from the AppServer/profiles/*profileName*/properties directory and put it in a thin client directory of your choice.

  com.ibm.CORBA.validateBasicAuth=false
  com.ibm.CORBA.securityServerHost=localhost
  com.ibm.CORBA.securityServerPort=<bootstrap port>
- Copy the ssl.client.props file from the AppServer/profiles/*profileName*/properties directory and put it in a thin client directory of your choice. This file contains the user.root property. You must modify the value to your thin client directory. This file refers to a key store, copy it from you application server or create new one and import the application server certificate.

- Copy the orb.properties file from the AppServer/properties directory and put it in a thin client directory of your choice. On Unix systems the file must be placed in the home of broker user or create a link from the file to the home.

For more information refer in the WebSphere® Application Server information center the article Using the Administration Thin Client.

The example above is changed in the Java code of the Construct Input node only. To see the changes read the following listing. The difference is a password and a virtual machine argument tag. This enables the code to handle the secured WebSphere® Application Server.

```
// set the credentials for the call
MbElement credentialElement = mbElement.createElementAsLastChild(MbElement.TYPE_NAME, "credentials", null);
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "userId", "admin");
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "password", "Test4now");
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "jmxUri", "service:jmx:rmi://localhost:2811/jndi/JMXConnector");
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "providerUri", "corbaloc:iiop://localhost:2811");
credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "initialContextFactory", "com.ibm.websphere.naming.WsnInitialContextFactory");
MbElement vmargsElement = credentialElement.createElementAsLastChild(MbElement.TYPE_NAME, "vmargs", null);
MbElement vmargElement = vmargsElement.createElementAsLastChild(MbElement.TYPE_NAME, "vmarg", null);
vmargElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "name", "com.ibm.SSL.ConfigURL");
vmargElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "value", "file:///home/jbenke/workspace/MyPlugins/BrokerThinClient/properties/ssl.client.props");
vmargElement = vmargsElement.createElementAsLastChild(MbElement.TYPE_NAME, "vmarg", null);
vmargElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "name", "com.ibm.CORBA.ConfigURL");
vmargElement.createElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "value", "file:///home/jbenke/workspace/MyPlugins/BrokerThinClient/properties/sas.client.props");
```

JMX Adapter Node Message Format

The root element of the message is jmxCalls it can handle unbounded number of independent JMX calls against a application server.

```
<xs:element name="jmxCalls">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="jmxCall" minOccurs="0" maxOccurs="unbounded" type="jmxCall" />
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

JmxCall defines the request itself with one credentials and one querystring element. The credentials contains the connection information and the querystring is the JMX request to fulfill. At the output terminal a result is added to the jmxCall structure.

```
<xs:complexType name="jmxCall">
        <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="1" name="credentials" type="credentials" />
                <xs:element minOccurs="1" maxOccurs="1" name="queryString" type="queryString" />
                <xs:element minOccurs="0" maxOccurs="1" name="result" type="result" />
        </xs:sequence>
</xs:complexType>
```

The credentials structure contains different elements to connect the JMX Interface of the application server.

- userId specify the user to access
- password of the user
- jmxUri is the URL to access the JSR160 interface of the application server
- providerUri is the URL to get the Initial Context of the application server.
- initialContextFactory is the Java Class Name for the Initial Context Factory of the application server.
- The element vmargs contains values to set runtime properties of the execution group virtual machine.

For a secured application server you need password and vmargs as shown in the example above.

```
<xs:complexType name="credentials">
	<xs:sequence>
		<xs:element minOccurs="1" maxOccurs="1" name="userId" type="xs:string" />
		<xs:element minOccurs="0" maxOccurs="1" name="password" type="xs:string" />
		<xs:element minOccurs="1" maxOccurs="1" name="jmxUri" type="xs:string" />
		<xs:element minOccurs="0" maxOccurs="1" name="providerUri" type="xs:string" />
		<xs:element minOccurs="0" maxOccurs="1" name="initialContextFactory" type="xs:string" />
		<xs:element minOccurs="0" maxOccurs="1" name="vmargs" type="vmargs" />
	</xs:sequence>
</xs:complexType>
```

A vmarg element has two attributes name and value which is contained in the vmargs. The attributes are runtime properties for the execution group virtual machine.

```
<xs:complexType name="vmargs">
	<xs:sequence>
		<xs:element minOccurs="1" maxOccurs="unbounded" name="vmarg" type="vmarg" />
	</xs:sequence>
</xs:complexType>

<xs:complexType name="vmarg">
	<xs:attribute name="name" use="required" />
	<xs:attribute name="value" use="required" />
</xs:complexType>
```

The querystring element has an attribute value which defines the number of objects to select. For example, WebSphere:type=ListenerPort,name=MY_TEST,* queries all listener ports with the name MY_TEST and WebSphere:type=ListenerPort,* queries all listener ports from the connected application server . With this attribute you can select the amount of objects to work on. The operation defines the method to call and the attributes the jmx object to get.

```
<xs:complexType name="queryString">
	<xs:sequence>
		<xs:element minOccurs="0" maxOccurs="1" name="operation" type="operation" />
		<xs:element minOccurs="0" maxOccurs="1" name="attributes" type="attributes" />
	</xs:sequence>
	<xs:attribute name="value" type="xs:string" use="required" />
</xs:complexType>
```

The operation element has one attribute value and unbounded number of arguments. Value defines the name of JMX operation and arguments contains elements which refers to the following structures.

```
<xs:complexType name="operation">
	<xs:sequence>
		<xs:element minOccurs="1" maxOccurs="1" name="arguments" type="arguments"></xs:element>
	</xs:sequence>
	<xs:attribute name="value" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="arguments">
	<xs:sequence>
		<xs:element minOccurs="0" maxOccurs="unbounded" ref="parameterType" />
	</xs:sequence>
</xs:complexType>
```

We have mainly two different types of arguments a XML representation of Java objects and a base64 encoding style.

```
<xs:element name="parameterType" abstract="true" />
```

The XML representation is possible for java.lang.String, java.util.Hashtable, java.util.Vector currently (only java.lang.String is in the documentation). If more types are needed the XSD has to be extended. The structure must match the output of the java.beans.XMLEncoder.

```
<xs:element name="java.lang.String" substitutionGroup="parameterType">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="java" minOccurs="0" maxOccurs="1">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="object">
							<xs:complexType>
								<xs:sequence>
									<xs:element name="string"
minOccurs="0" maxOccurs="1" type="xs:string" />
								</xs:sequence>
								<xs:attribute name="class" use="required"
fixed="java.lang.String" />
							</xs:complexType>
						</xs:element>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

To use the base 64 style encoding as follows. Serialize the Java object and decode the result with base64 encoder (Use the com.ibm.misc.BASE64Encoder). The result string is the value for the serialized object element.

```
</xs:element><xs:element name="serializedObject" substitutionGroup="parameterType">
	<xs:complexType>
		<xs:sequence>
			<xs:element minOccurs="1" maxOccurs="1" name="object" />
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

The result element contains a base64 encode serialized java object which represents the return of the jmx operation and the value of attributes. How to analyze result is shown in a detailed example above (Java Code). Use the deliver function `JMXHelper.getResultVector` to get the result of a request.

```
<xs:complexType name="result">
        <xs:sequence>
                <xs:any minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
</xs:complexType>
```