HP OpenView Smart Plug-in User Defined Metrics

For the HP-UX and Solaris HP OpenView Management Servers

Software Version: 04.20

User Guide

Document Release Date: January 2008
Software Release Date: November 2006

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

HP does not guarantee that it will or can fix any problems found with JMB on JMX Compliant Mbean server deployments. Problems found with the deployment and working of the JMB on a non WebLogic/WebSphere environment must be routed as a consulting assignment and will not be considered a support call for the JMB.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

UNIX® is a registered trademark of The Open Group.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Documentation Updates

This manual's title page contains the following identifying information:

- Software version number, which indicates the software version
- Document release date, which changes each time the document is updated
- Software release date, which indicates the release date of this version of the software

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

http://ovweb.external.hp.com/lpe/doc_serv/

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

You can visit the HP OpenView Support web site at:

www.hp.com/managementsoftware/support

HP OpenView online support provides an efficient way to access interactive technical support tools. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:

www.managementsoftware.hp.com/passport-registration.html

Contents

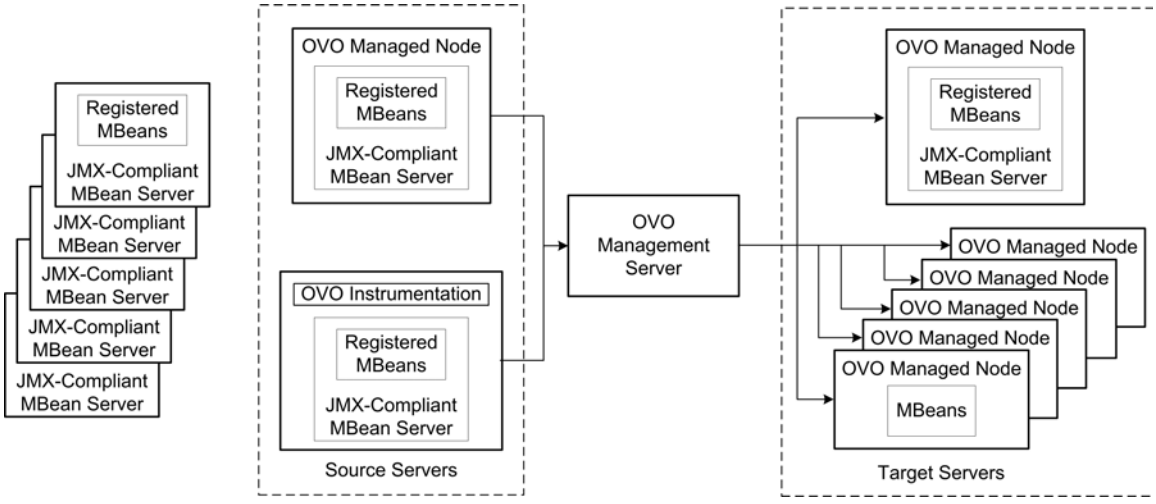
1 Overview

User defined metrics (UDMs) are metrics created by the user to gather data from application MBeans registered in a BEA WebLogic MBean server or PMI counters if you are running a WebSphere application server. You can create UDMs by using the JMX Metric Builder (JMB), JMB Plug-in for Eclipse, or by editing the metrics definition XML file (also referred to as the UDM file).

To monitor your applications using HP OpenView Operations (OVO), you configure your MBean server environment and create templates to monitor and collect the data generated by the UDMs.

The MBean Server Environment

Using OVO and the HP OpenView Smart Plug-In for WebSphere Application Server (WBS SPI) or HP OpenView Smart Plug-In for BEA WebLogic Server (WLS SPI), the OVO management server can monitor servers whose MBeans are registered in a WebLogic or WebSphere MBean server. The OVO management server must be configured to gather MBean data (also referred to as metadata) from source servers and to monitor target servers using UDMs.



Source servers are systems on which the WebSphere or WebLogic MBean server resides. The OVO instrumentation must be distributed to the source servers (the OVO management server may not monitor them). The source servers can be an MBean staging area or development server.

The OVO management server collects MBean data from the source servers. Select a subset of your MBean servers (those that have a representative set of MBeans registered) to be your source servers.

Target servers are systems that are monitored by the OVO management server. The target server can be a production server. Alarms, graphs, and reports are generated by UDMs based on MBeans registered in the WebLogic or WebSphere MBean server.

Using the WebLogic or WebSphere MBean Server

Both the WebLogic and WebSphere application servers (WebSphere application server version 5.0 or higher) include a built-in MBean server. Additional tasks are required to create UDMs such as, installing the WASSPI-UDM-Bldr software bundle, configuring the WLS SPI or WBS SPI to collect the MBean data, and using the JMX Metric Builder (an application that helps you create UDMs and browse MBeans). These tasks are included in chapters 2 and 3 of this guide.

Using a JMX-Compliant MBean Server

If you are using a JMX-compliant MBean server that is not part of the WebLogic or WebSphere application server, you must install and configure the JMX connector before you can create UDMs using the JMB. You must also install the WBS SPI or WLS SPI which installs the collector that allows OVO to collect the MBean data.

To use the JMX connector, you should be familiar with the following topics:

- the standard MBeanServer interface
- MBean registration
- m-let service
- JAAS (Java Authentication and Authorization Service)

  JAAS is applicable if MBean server access requires either or both authentication and authorization.

OVO and Other Components

Additional components are required to create UDMs. Some of these components require additional configuration.

The JMX Metric Builder

The JMX Metric Builder (JMB) is an application integrated with OVO used to create UDMs that gather data from application MBeans registered in the WebLogic or WebSphere MBean server. You can edit the UDM file by mapping MBeans to UDMs, validate metric IDs, and create UDMs that conform to the metric definitions DTD. You can also use the JMB to browse MBeans on a configured MBean server and generate OVO templates. For more information about using the JMB, refer to the online help.

MBean data is obtained from a cache on the OVO management server. You must run the Gather MBean Data application to gather the MBean data that is stored in the cache on the OVO management server. For more information about the Gather MBean Data application and configuration requirements, refer to The Gather MBean Data Application on page 14.

The JMB is installed with the WASSPI-UDM-Bldr software bundle. For more information, refer to Installing the WASSPI-UDM-Bldr Software Bundle on page 18.

The JMX Metric Builder Plug-in for Eclipse

The JMX Metric Builder Plug-in for Eclipse (JMB Plug-in for Eclipse) is the same application as the JMB but is run independently from the OVO environment (the JMB Plug-in for Eclipse is launched from Eclipse, not OVO). The JMB Plug-in for Eclipse includes the same features as the JMB and can also test UDMs (For more information about these features, refer to the online help).

MBean data is obtained directly from the application server (currently, the JMB Plug-in for Eclipse only supports the BEA WebLogic Application Server). This allows a developer to create UDMs and templates outside of the OVO environment. UDMs and templates generated by the JMB Plug-in for Eclipse must be copied to the OVO management server and deployed to managed nodes.

The JMB Plug-in for Eclipse is downloaded from the OVO management server (you must install the WASSPI-UDM-Bldr software bundle (for more information, refer to Installing the WASSPI-UDM-Bldr Software Bundle on page 18) or from the HP Developer's Resource web page (**http://devresource.hp.com/drc/resources/jmbdwnld/index.jsp**).

For more information about system requirements, installing the JMB Plug-in for Eclipse, and integrating with OVO, refer to the *Getting Started* guide (`JMBPlugin_GettingStarted.pdf`). On the OVO management server, extract the guide from one of the following files:
`/opt/OV/jmb/JMB-Developer/JMBPlugin-Windows.zip` or
`/opt/OV/jmb/JMB-Developer/JMBPlugin-HPUX.zip`.

The JMB Plug-in for Eclipse only supports the BEA WebLogic Application Server.

The Gather MBean Data Application

The Gather MBean Data application gathers MBean information from selected managed node(s) and allows you to gather the MBean data at any time. The `COLLECT_METADATA` property must be set on the managed node for the collection to occur. The tasks required to set this property are included in the configuration chapter of this guide. For more information, refer to Gather MBean Data on page 88.

The Gather MBean Data application is installed with the WASSPI-UDM-Bldr software bundle. For more information, refer to Installing the WASSPI-UDM-Bldr Software Bundle on page 18.

Metric and Collector Templates

Metric and collector templates are monitor templates you must create before you can successfully monitor the target servers in your MBean server environment.

A metric template monitors performance levels of a metric by defining threshold conditions for the metric. Within a metric template, you can also define the message text sent to the OVO message browser when the threshold is exceeded, the actions to execute, and the instruction text that appears.

A collector template specifies the collection interval of one or more metric templates. That is, it determines how often data is collected for a metric or group of metrics and compared to the threshold condition.

Both templates must be defined and distributed to the target servers (these tasks are included in the UDM creation chapter of this guide).

SPI Applications and the JMX Connector

If you are using a JMX-compliant MBean server that is not part of the WebLogic or WebSphere application server, you must install either the WBS SPI or WLS SPI. When you install the SPI software, three application groups are installed: WBSSPI/WLSSPI Admin, WebSphere/WebLogic, and WBSSPI/WLSSPI Reports.

The following WBSSPI/WLSSPI Admin applications can be run even if you are not managing a WebSphere or WebLogic application server (applications in the Admin applications group not listed here cannot be run successfully):

- Config WBSSPI/WLSSPI
- Self-Healing Info
- Start/Stop Monitoring
- Start/Stop Tracing
- Verify
- View Error File
- View Graphs

For more information about these applications, refer to the WBS SPI or WLS SPI configuration guide.

PMI Counters

If you are creating UDMs based only on PMI counters, you must install the WBS SPI and WASSPI-UDM-Bldr software bundle (to install the Deploy UDM application). For more information, see Installing the WBS SPI/WLS SPI Software on page 17 and Installing the WASSPI-UDM-Bldr Software Bundle on page 18.

After you have configured the WBS SPI, no additional configuration is needed to create UDMs based on PMI counters.

You cannot use the JMB to create or edit UDMs based on PMI counters. Do not open the `/opt/OV/wasspi/udm/conf/wbs/UDMMetrics-sample.xml` file in the JMB. If you open this file in the JMB, all your PMI counter metrics are converted to hidden metrics (hidden metrics can only be used to calculate other metrics; they cannot be used as alarming, graphing, nor reporting metrics).

WBS SPI and WLS SPI Configuration Guides

The latest versions of the WBS SPI and WLS SPI configuration guides can be found online at:

http://ovweb.external.hp.com/lpe/doc_serv/

1. In the Select Product list, click either **smart plug-ins for BEA WebLogic server** or **smart plug-ins for WebSphere**.
2. In the Select Version list, click **04.02.000**.
3. In the Select OS list, click **HP-UX**.
4. In the Select the Product Manual Title list, click **configuration guide**.
5. Click **Open** to view the guide.

2 Installation, Upgrade, and Removal

Before you can develop UDMs using the JMX Metric Builder (JMB), you must install the SPI software and WASSPI-UDM-Bldr software bundle. Depending on your MBean server environment, additional software (the JMX connector) may need to be installed.

Built-in MBean server requirements: If you are using the MBean server that is built into the WebLogic or WebSphere application server, you must install the following software:

- WBS SPI or WLS SPI
- WASSPI-UDM-Bldr bundle

JMX-compliant MBean server requirements: If you are using a JMX-compliant MBean server that is not part of the WebLogic or WebSphere application server, you must install the following software:

- WBS SPI or WLS SPI
- WASSPI-UDM-Bldr bundle
- JMX connector

This chapter includes instructions for installing the WBS SPI/WLS SPI software, installing and removing the WASSPI-UDM-Bldr software bundle, installing and removing the JMX connector, upgrading the JMX connector, and upgrading custom collector templates that monitor non-WebLogic or non-WebSphere MBean servers.

Installing the WBS SPI/WLS SPI Software

Complete SPI software installation information is available in the SPI configuration guide.

For an HP-UX 11.0 management server, enter:

```
swinstall -s /cdrom/OV_DEPOT/11.0HPUX.sdtape WBSSPI or
swinstall -s /cdrom/OV_DEPOT/11.0HPUX.sdtape WLSSPI
```

For a Solaris management server, enter:

```
swinstall -s /cdrom/OV_DEPOT/SOLARIS.sdtape WBSSPI or
swinstall -s /cdrom/OV_DEPOT/SOLARIS.sdtape WLSSPI
```

If you are using the MBean server that is built into the application server, you must configure the WBS SPI or WLS SPI software. For more information, refer to the WLS SPI or WBS SPI configuration guide.

To remove the SPI software, refer to the SPI configuration guide.

Installing the WASSPI-UDM-Bldr Software Bundle

The following examples show the command line usage of swinstall. For HP-UX systems, you can also use the graphical user interface (GUI), but the GUI method is not covered.

For an HP-UX 11.0 management server, enter:

```
swinstall -s /cdrom/OV_DEPOT/11.0HPUX.sdtape WASSPI-UDM-Bldr
```

For a Solaris management server, enter:

```
swinstall -s /cdrom/OV_DEPOT/SOLARIS.sdtape WASSPI-UDM-Bldr
```

The WASSPI-UDM-Bldr software bundle includes the following:

Item		Description	Location
Applications	Deploy UDM	Deploys the UDM file from the management server to the selected managed node(s)	JMX Metric Builder/ WBSSPI or JMX Metric Builder/ WLSSPI application group
	Gather MBean Data	Gathers MBean information from the selected managed node(s)	
	JMX Metric Builder	Launches the JMB	
	UDM Graph Enable/Disable	Starts/stops data collection for UDM graphs	
Archive Packages	`RMI_ConnectorPkg.tar` `RMI_ConnectorPkg.zip`	JMX connector archive packages	`/opt/OV/wasspi/wbs/jmx/` or `/opt/OV/wasspi/wls/jmx/`

Installing the JMX Connector

If you are using MBeans registered with the WebLogic or WebSphere MBean server, you do *not* need to install and configure the JMX connector. The WLS SPI/WBS SPI provides this functionality.

The JMX connector allows access to MBeans registered in other non-WebLogic or non-WebSphere MBean servers.

The instructions that follow are for the WLS SPI. If you are installing the WBS SPI, replace any occurrence of WLSSPI with WBSSPI and of wls with wbs.

1. The JMX Connector archive packages are installed with the WASSPI-UDM-Bldr software bundle. Copy one of the following archive packages to all source and target servers:

   `/opt/OV/wasspi/wls/jmx/RMI_ConnectorPkg.tar` (UNIX)
   `/opt/OV/wasspi/wls/jmx/RMI_ConnectorPkg.zip` (Windows)

2. On each server, extract the files from the archive package. The files do not have to be placed in any specific directory.

   The following files are extracted from the archive package.

   - `JMX_RMI_Connector.jar` - The JMX connector.
   - `JMXRMIConnectorConfig` - Example JMX connector configuration file.
   - `OVRMIConnectorMbean.mlet` - Example m-let file.
   - `OVRMIConnectorTrace.tcf` - Trace configuration file.

   For more information about these files, see Other JMX-Compliant MBean Server on page 30.

Upgrading the JMX Connector

It is recommended that you upgrade the connector MBean. To upgrade the connector MBean, unregister the current connector MBean and register the new one.

Upgrading Templates that Monitor Non-WebLogic or Non-WebSphere MBean Servers

If you created custom collector templates that monitor non-WebLogic or non-WebSphere MBean servers and have upgraded the SPI to version 04.20.000, you must upgrade the custom collector templates:

1. Open the Message Source Templates window.
2. Open the template group in which the custom collector templates reside.
3. For each custom collector template:
   a. Click **Modify**.
   b. Append **`-type ovrmi`** to the end of the Monitor Program or MIB ID text box.
   c. Click **OK**.
4. Distribute the templates:
   a. Open the Node Group Bank window and select all the managed nodes on which you are monitoring non-WebLogic or non-WebSphere MBean servers.
   b. From the Actions menu, select **Agents** → **Install/Update SW & Config**.
   c. In the Target Nodes section, select the **Nodes in List Requiring Update** radio button.
   d. In the Install/Update Software and Configuration window, check the **Templates** check box.
   e. Select **Force Update**.
   f. Click **OK**.

   The following message appears in the message browser:

   ```
   The following configuration information was successfully distributed:
   Templates
   ```

Removing the WASSPI-UDM-Bldr Software Bundle and OVO Components

Complete these tasks only if you no longer wish to create nor use UDMs and do not want to use the JMX Metric Builder.

Complete the tasks in the order listed:

- Task 1: Remove Software from the Management Server
- Task 2: Remove Software from the Node Group and Managed Nodes
- Task 3: Delete Custom Templates and Template Groups
- Task 4: Delete Applications
- Task 5: Delete Custom Message and Node Groups

Task 1: Remove Software from the Management Server

1. Open a terminal window and log on as root.
2. In the terminal window, enter the following:

   `/usr/sbin/swremove WASSPI-UDM-Bldr`

The swremove command removes the files from the file system only. Your customized templates are still in the OVO data repository and must be deleted manually. Before the templates can be deleted, they (and the software) must be de-assigned from the managed nodes (see Task 2: Remove Software from the Node Group and Managed Nodes).

Task 2: Remove Software from the Node Group and Managed Nodes

1. Open the Node Bank and from the Actions menu select **Agents → Assign Templates**.
2. Select all custom node groups and all managed nodes to which your customized templates have been assigned.
3. Click **Remove nodes/groups**.
4. Open the Node Group Bank and select all custom node groups.
5. From the Action menu select **Install/Update SW & Config** and check the following check boxes:
   - Templates
   - Actions
   - Monitors
   - Commands
6. Select **Nodes in List**.
7. Select **Force Update**.
8. Click **OK** to remove the Templates, Actions, Commands and Monitors from the managed nodes. The following message appears in the Message Browser:

   ```
   The following configuration information was successfully distributed:
   Templates Actions Commands Monitors
   ```

Task 3: Delete Custom Templates and Template Groups

Delete all custom templates and template groups (for metrics that you no longer wish to collect).

1. Open the Message Source Templates window and double-click the custom template group
2. Press **SHIFT** and click to select all templates and template groups.
3. Click **Delete from All...**. The following message appears:

   `Do you really want to delete the template(s)?`
4. Click **YES**.
5. If there are additional custom templates or template groups, repeat steps 2 -4 until you have deleted all custom templates and template groups.
6. Go up one level and delete the custom template group.

Task 4: Delete Applications

Unlike templates, applications can be removed in a single step.

1. Open the Application Bank.
2. Right-click the JMX Metric Builder application group and click **Delete**.

   The following message appears:

   `Do you really want to delete the application group?`
3. Click **Yes**.

Task 5: Delete Custom Message and Node Groups

1. From the Window menu select **Message Group Bank**.
2. In the Message Group Bank window right-click the custom group and click **Delete**.
3. Repeat for any other custom group.
4. From the Window menu select **Node Group Bank**.
5. In the Node Group Bank window right-click each custom group and click **Delete**.

Removing the JMX Connector

The instructions that follow are for the WLS SPI. If you are removing the WBS SPI, replace any occurrence of WLSSPI with WBSSPI and of wls with wbs.

1. Remove the archive packages from the management server:

   ```
   rm /opt/OV/wasspi/wls/jmx/RMI_ConnectorPkg.tar
   rm /opt/OV/wasspi/wls/jmx/RMI_ConnectorPkg.zip
   ```

2. Remove the archive packages from all source and target servers:

   `/opt/OV/wasspi/wls/jmx/RMI_ConnectorPkg.tar` (UNIX)
   `/opt/OV/wasspi/wls/jmx/RMI_ConnectorPkg.zip` (Windows)

3. Remove the following files from all source and target servers:

   ```
   JMX_RMI_Connector.jar
   JMXRMIConnectorConfig
   OVRMIConnectorMbean.mlet
   OVRMIConnectorTrace.tcf
   ```

3 Configuration

Before you can develop UDMs using the JMX Metric Builder (JMB), your environment must be configured so that MBean information (metadata) is collected from your MBean server(s).

To configure your environment, you must complete the following tasks:

- Register Your Custom MBeans (optional)
- Configure your MBean server environment (see page 26)
- Complete additional configuration tasks (see page 38)

Register Your Custom MBeans

Before configuring your MBean server environment, register your custom MBeans (this task is optional). However, custom MBeans must be registered in the WebLogic or WebSphere MBean server if you want to monitor and collect its data.

If you are using the WebLogic MBean server, the Name attribute is used to identify its MBeans. If your MBean is a multi-instance MBean, then each MBean instance must have a unique value in its Name attribute. For example, WebLogic's ServletRuntime MBeans are multi-instance because a ServletRuntime MBean is instantiated by WebLogic for each deployed servlet. The Name attribute of the MBean identifies the servlet that the MBean is monitoring. If the Name attribute is not provided, then the full ObjectName is used as the instance identifier.

If you are using the WebSphere MBean server, the mbeanIdentifier ObjectName key property is used to identify its MBeans. If your custom MBean is a multi-instance MBean, then each MBean instance must have a unique value in its mbeanIdentifier ObjectName key property. If the mbeanIdentifier ObjectName key property is not provided, then the full ObjectName is used as the instance identifier.

For any other JMX-compliant MBean server, the full ObjectName is used as the instance identifier.

Refer to your JMX-compliant server documentation for information about creating and registering MBeans.

JMX specifications are located at **http://java.sun.com/products/JavaManagement/reference/docs/index.html**

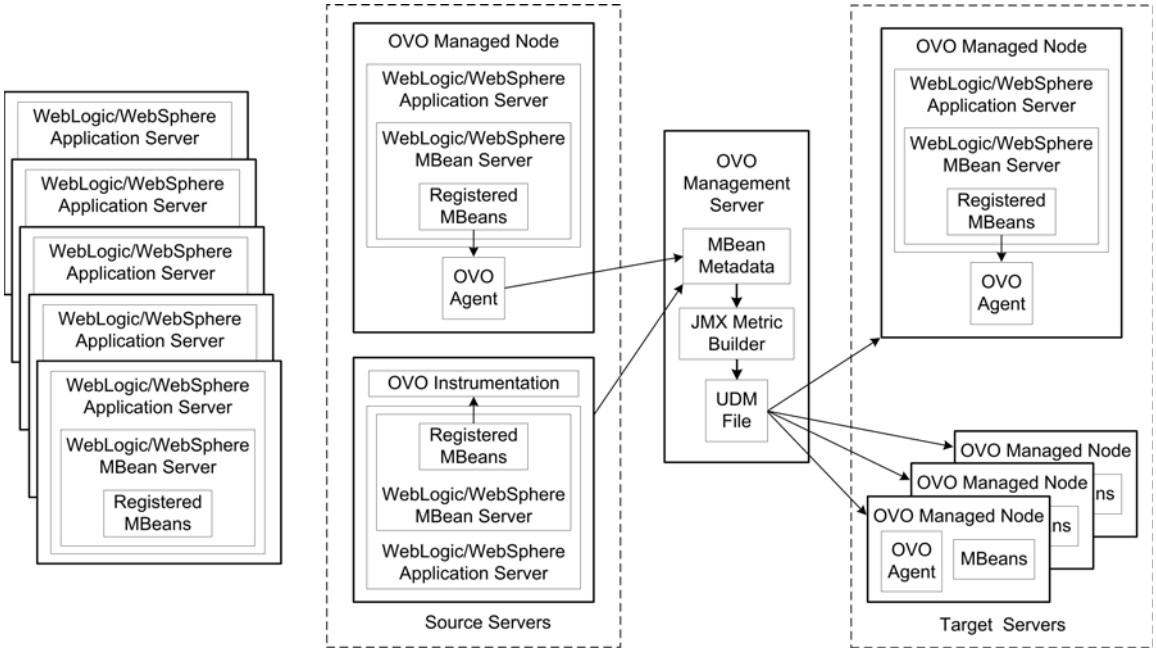
MBean Server Environment Configuration

The node on which the WebLogic or WebSphere MBean server is running must be configured so that MBean information can be gathered. Configuration depends upon the type of MBean server you are using (complete the steps listed in the appropriate section):

- WebLogic/WebSphere MBean Server
- JBoss MBean Server
- Other JMX-Compliant MBean Server

WebLogic/WebSphere MBean Server

If you are using the MBean server that comes with the WebLogic or WebSphere application server, your MBean server environment may look similar to the following:



To configure this MBean server environment, complete the following steps:

1 Configure the WLS SPI/WBS SPI software on the source and target servers. For more information, refer to chapter 3 of the WLS SPI or WBS SPI configuration guide.

   If you do not want to monitor the source server, do not distribute the SPI templates to the source server during the SPI configuration process.

2 Set the COLLECT_METADATA server property of the source server to ON and the JMB_JAVA_HOME property to an installation of Java version 1.4.1 or higher (this example shows the steps using the WLS SPI; if you have installed the WBS SPI, simply change any occurrence of WLSSPI to WBSSPI):

   a At the OVO console, select the source server in the Node Bank window.

   b From the Window menu, select **Application Bank**.

   c In the Application Bank window select **WLSSPI → WLSSPI Admin**.

   d Double-click **Config WLSSPI**.

e The Introduction window opens. This window contains brief information about the configuration editor.

   Click **Next**.

f From the configuration editor, set the COLLECT_METADATA property to ON for the source server and the JMB_JAVA_HOME property to an installation of Java version 1.4.1 or higher for the management server. For more information about using the configuration editor, refer to Appendix B of the WLS SPI or WBS SPI configuration guide.

g Click **Next** to save the change and exit the editor.

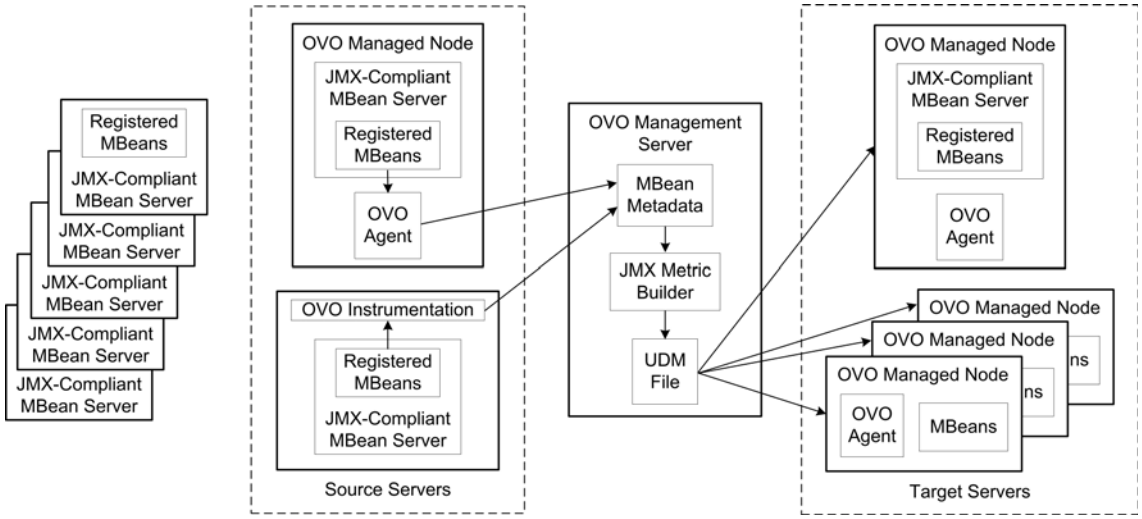
h The Confirm Operation window opens. Click **OK**.

> If you click **Cancel** and made changes to the configuration, those changes remain in the configuration on the management server. To make the changes to the selected managed nodes' configuration, you must select those nodes in the Node Bank window, start the Config WLSSPI application, click **Next** from the configuration editor, and then click **OK**.

3 Complete the Additional Configuration on page 38.

JBoss MBean Server

If you are using the JBoss MBean server, your MBean server environment may look similar to the following:



To configure the JBoss MBean server environment, do the following:

1 Register the JMX connector.

   a Create a `jboss-service.xml` file. In the `META-INF` directory of your service archive (SAR) directory, create a `jboss-service.xml` file (if the `META-INF` directory does not exist, create it). For example, create <*jboss_install*>`/server/default/deploy/OVJMXConnector.sar/jboss-service.xml`.

Enter the following in the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <classpath codebase=”lib” archives=”*”/>

<!-- This is the JMX Connector in JMX_RMI_Connector.jar -->
  <mbean code=”com.hp.openview.wasspi.connector.rmi.RemoteMBeanServerImpl”
   name=”OpenView:type=connector,protocol=RMI”>
  </mbean>
</server>
```

b Edit the <*jboss_install*>/server/default/deploy/properties-service.xml file. If you are running JBoss 4.0.0, you may have to copy this file from the <*jboss_install*>/server/all/deploy/ directory.

Update the SystemProperties MBean and set the com.hp.openview.jmx.connector.config.file property to the fully qualified path name of the JMXRMIConnectorConfig file.

The file may look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE server>
<!-- $Id: properties-service.xml,v 1.5 2003/08/27 04:34:12
patriotlburke Exp $ -->

<server>

<!-- ===================================== -->
<!-- PropertyEditorManager Service -->
<!-- ===================================== -->

<!--
   | Allows access to the PropertyEditorManager, which is used when
setting
   | MBean attribute values from configuration.
-->

<mbean code="org.jboss.varia.property.PropertyEditorManagerService"
   name="jboss:type=Service,name=PropertyEditorManager">

  <!--
     | Register an editor for each of the type_name=editor_type_name
listed
     | in properties file style convention.

  <attribute name="Editors">
    java.net.URL=my.project.editors.URLPropertyEditor
  </attribute>

  -->

</mbean>

<!-- ===================================== -->
<!-- System Properties Service -->
<!-- ===================================== -->
```

```
<!--
   | Allows rich access to system properties.
 -->

<mbean code="org.jboss.varia.property.SystemPropertiesService"
   name="jboss:type=Service,name=SystemProperties">

  <!--
     | Load properties from each of the given comma seperated URLs

  <attribute name="URLList">
    http://somehost/some-location.properties,
    ./conf/somelocal.properties
  </attribute>

   -->

  <!--
     | Set raw properties file style properties.
  <attribute name="Properties">

    my.project.property=This is the value of my property
    my.project.anotherProperty=This is the value of my other property
  </attribute>

   -->

  <attribute name="Properties">
  com.hp.openview.jmx.connector.config.file=
C:/jmx/jboss-4.0.1sp1/server/default/deploy/OVJMXConnector.sar/
JMXRMIConnectorConfig
  </attribute>
</mbean>
</server>
```

c On the source server, edit the following properties in the `JMXRMIConnectorConfig` file:

— `com.hp.openview.jmx.connector.bind.name`

Required. The name of the JBoss MBean server to which the JMX connector binds. When you configure the SPI to use the JMX connector, use this same value when setting the NAME property.

— `com.hp.openview.jmx.connector.rmi.registry.port`

Required. The port on which the JBoss MBean server listens. When you configure the SPI to use the JMX connector, use this same value when setting the PORT property.

— `com.hp.openview.jmx.connector.trace.config.file`

Required. The name and location of the `OVRMIConnectorTrace.tcf` file.

d Deploy the JMX connector as a service. Copy the following files to a service archive directory such as `<jboss_install>/server/default/deploy/OVJMXConnector.sar/`:

```
JMX_RMI_Connector.jar
JMXRMIConnectorConfig
META-INF/jboss-service.xml
```

The `com.hp.openview.jmx.connector.config.file` property in the `jboss-service.xml` file should be set to the fully qualified path name of the `JMXRMIConnectorConfig` file.

e Implement a JMXAuthenticator (optional) for JBoss 4.0 security considerations. For more information, see Appendix D, Authenticating with JBoss UsersRolesLoginModule.

f Redeploy the SAR.

If you restart JBoss, edit `properties-service.xml`, then edit `JMXRMIConnectorConfig`, and/or update `JMX_RMI_Connector.jar`, you must redeploy the SAR.

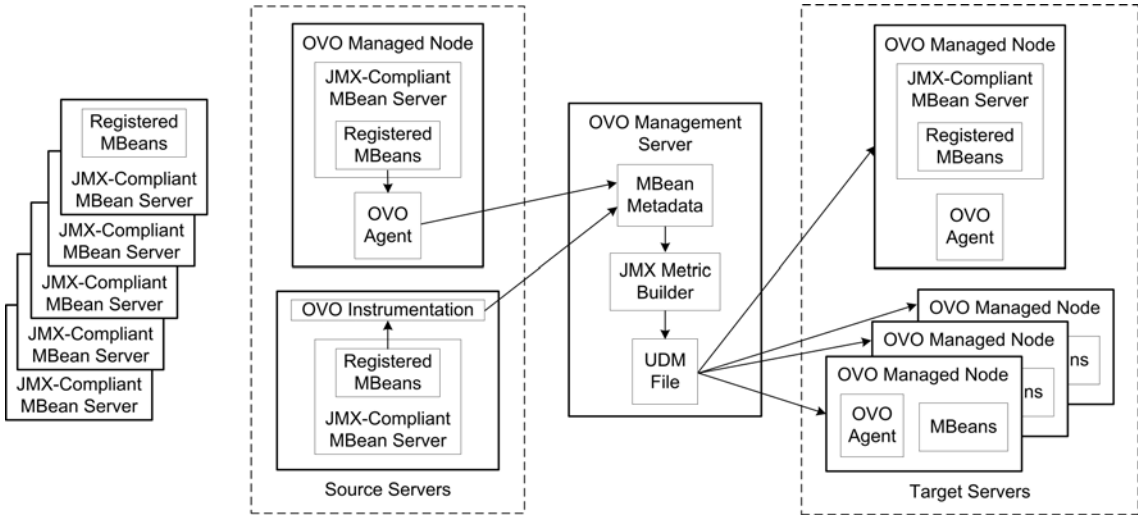
To redeploy the SAR, move the SAR directory out of and then back into the *<jboss_install>*`/server/default/deploy/` directory. For example, type the following:

```
cd <jboss_install>/server/default/deploy/
mv OVJMXConnector.sar ..
mv ../OVJMXConnector.sar .
```

2 To complete your JBoss MBean server environment configuration, go to Additional JMX-Compliant MBean Server Configuration on page 33.

Other JMX-Compliant MBean Server

If you are using a JMX-compliant MBean server which is not the MBean server that comes with the WebLogic or WebSphere application server, your MBean server environment may look similar to the following:



To configure the JMX-compliant MBean server environment, do the following:

1 On the source server, edit the `JMXRMIConnectorConfig` file.

A sample `JMXRMIConnectorConfig` file is included in the archive package.

If you are registering the JMX connector programmatically and are setting the configuration properties programmatically, you do not have to edit the `JMXRMIConnectorConfig` file. That is, if you are using the `Properties` argument of the `RemoteMBeanServerImpl` constructor, you do not have to edit the `JMXRMIConnectorConfig` file.

If you are registering the JMX connector by using the m-let service MBean or if you are registering the JMX connector programmatically and are using a configuration file to set the configuration properties, edit the following properties in the `JMXRMIConnectorConfig` file:

- `com.hp.openview.jmx.connector.bind.name`

  Required. The RMI registry name to which the connector service binds. When you configure the SPI to use the JMX connector, use this same value when setting the NAME property.

- `com.hp.openview.jmx.connector.rmi.registry.port`

  Required. The port on which the RMI registry server listens. When you configure OVO to use the JMX connector, use this same value when setting the PORT property.

- `com.hp.openview.jmx.connector.authenticator`

  Optional. The full package name of a class implementing the `com.hp.openview.wasspi.connector.rmi.JMXAuthenticator` interface. This can also be any class-defining method signature: `public Subject authenticate` *(Object credentials)*. The `authenticate` method performs JAAS authentications on credentials supplied by the SPI collector. Credentials are in the form of a two element string array containing a login and password.

- `com.hp.openview.jmx.connector.authorization`

  Optional. Default: true. Only applicable when an authenticator is provided. If true, the JMX connector calls the MBean server methods as the authenticated JAAS subject. If authorization is performed by a wrapper MBeanServer, call the JMX connector method `setMBeanForwarder(MBeanServer mbs)` following registration which sets this property to false.

2 Register the JMX connector.

You can register the connector in one of the following ways:

- Use the m-let service MBean to register the JMX connector:

  A sample `m-let` file, `OVRMIConnectorMbean.mlet`, is included in the archive package.

  The following is an example of the m-let file:

```
<MLET
 CODE = com.hp.openview.wasspi.connector.rmi.RemoteMBeanServerImpl
 ARCHIVE = "JMX_RMI_Connector.jar"
 NAME = OpenView:type=Connector,protocol=rmi
>
<ARG TYPE = java.lang.String
 VALUE = <pathname>/JMXRMIConnectorConfig>
</MLET>
```

  In this example,

  — the `m-let` file and `JMX_RMI_Connector.jar` files are in the same directory.

  — all attributes, except NAME, are required. The NAME attribute is optional and, if omitted, is assigned an object name by the MBean server.

— the ARG attribute must be set to the fully qualified path name of the `JMXRMIConnectorConfig` file configured in step 1 on page 30.

If the `m-let` file and `JMX_RMI_Connector.jar` files are not in the same directory, configure the m-let CODEBASE attribute to the location of the `JMX_RMI_Connector.jar` file (include the fully qualified path name).

- Programmatically register the JMX connector:

> The RemoteMBeanServer is the RMI equivalent of the standard JMX MBeanServer interface. You should instantiate and register RemoteMBeanServerImpl (the implementation of RemoteMBeanServer)

The JMX connector can be registered programmatically in one of the following ways (examples are shown) in the MBean server:

— `public RemoteMBeanServerImpl()` - set the configuration properties in the `JMXRMIConnectorConfig` file and set the system property com.hp.openview.jmx.connector.config.file to the absolute location of the `JMXRMIConnectorConfig` file.

In this example, the `com.hp.openview.jmx.connector.config.file` property is either set when JVM is started or earlier in the application. The application uses `buildObjectName` (optional) allowing the JMX connector to specify its own object name.

```
// Create the MBean server
MbeanServer mbeanServer = MBeanServerFactory.createMBeanServer();

// Create the JMX connector for OVO
RemoteMBeanServerImpl remoteServer = new RemoteMBeanServerImpl();
ObjectName objName = remoteServer.buildObjectName();

//Register the JMX connector
mbeanServer.registerMBean(remoteServer, objName);
```

— `public RemoteMBeanServerImpl("<pathname>/JMXRMIConnectorConfig")` - set the configuration properties in the JMXRMIConnectorConfig file and pass the `JMXRMIConnectorConfig` file name as an argument.

```
// Create the MBean server
MbeanServer mbeanServer = MBeanServerFactory.createMBeanServer();

// Create the JMX connector for OVO, specifiying the fully qualified path name
// of the configuration file
RemoteMBeanServerImpl remoteServer = new
 RemoteMBeanServerImpl("C:/test/jmx/JMXRMIConnectorConfig");

// Register the JMX connector with an explicit object name
ObjectName objName = new ObjectName("mydomain:type=connector,protocol=rmi");
mbeanServer.registerMBean(remoteServer, objName);
```

— `public RemoteMBeanServerImpl(<ConfigurationProperties>)` - set the configuration properties programmatically.

```
// Create the MBean server
MbeanServer mbeanServer = MBeanServerFactory.createMBeanServer();

// Set configuration properties
Properties configProps = new Properties();

configProps.setProperty(RemoteMBeanServer.REGISTRY_BIND_NAME_PROPERTY,
"myMBeanServer");

configProps.setProperty(RemoteMBeanServer.REGISTRY_PORT_PROPERTY, "1102");

configProps.setProperty(RemoteMBeanServer.TRACE_CONFIG_PROPERTY,
"C:/test/jmx/OVRMIConnectorTrace.tcf");

// Optional JAAS authenticator
configProps.setProperty(RemoteMBeanServer.AUTHENTICATOR_PROPERTY,
"my.package.MyAuthenticator");

// Optional and only applicable when a JAAS authenticator is provided
configProps.setProperty(RemoteMBeanServer.AUTHORIZATION_PROPERTY,
"false");

// Create the JMX connector
RemoteMBeanServerImpl remoteServer = new
RemoteMBeanServerImpl(configProps);

//Register the JMX connector
ObjectName objName = new ObjectName("mydomain:type=connector,protocol=rmi");
mbeanServer.registerMBean(remoteServer, objName);
```

3 To complete your JMX-compliant MBean server environment configuration, go to Additional JMX-Compliant MBean Server Configuration on page 33.

Additional JMX-Compliant MBean Server Configuration

If you are using JBoss or any other JMX-compliant MBean server which is not the MBean server that comes with the WebLogic or WebSphere application server, complete the following tasks:

1 Set permissions.

If a security manager is installed on the JVM running the MBean server, you may need to set the following permissions in your security policy file:

Item \ Permissions	Read	Write	Listen	Resolve
Configuration file directory	✓			
Property name TRACE_CFG_FILE	✓	✓		
All non-system ports on localhost (ports > 1024)			✓	✓

2 Configure the source and target servers to be managed nodes.

From the OVO management server, do the following (this example shows the steps using the WLS SPI; if you have installed the WBS SPI, simply change any occurrence of WLSSPI to WBSSPI).

a From the OVO console, open the Node Bank window and select the source and target servers.

b From the Node Bank's Actions menu select **Agents → Install/Update SW & Config**.

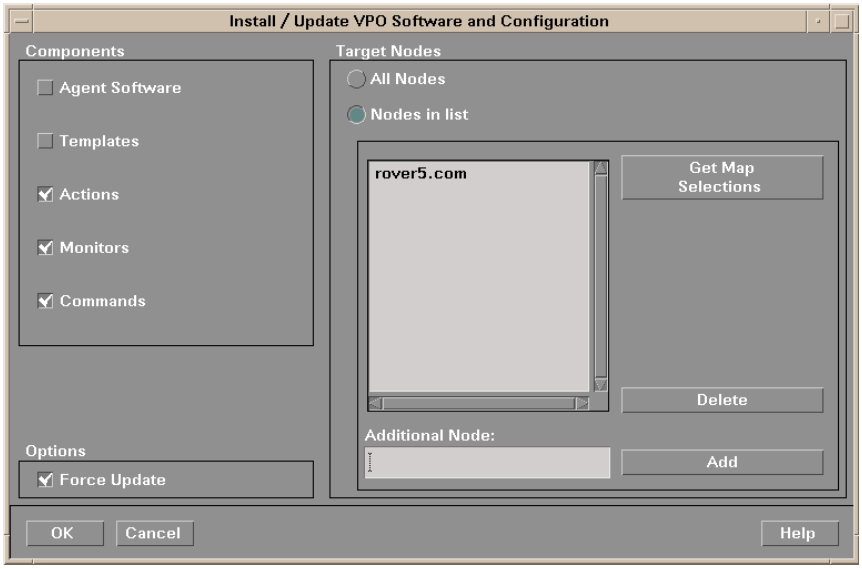
c In the Install/Update OVO Software and Configuration window select the following component check boxes:

— Actions

— Monitors

— Commands

Using this dialog, deploy program components to the managed node(s).



d Select the **Force Update** check box.

e Click **Nodes in list**. Upon completion, the following message appears in the Message Browser for each managed node:

```
The following configuration information was successfully distributed:
Actions Commands Monitors
```

3 Configure the source server.

To configure the JMX collector in OVO, use the SPI configuration editor. From the OVO management server, do the following (this example shows the steps using the WLS SPI; if you have installed the WBS SPI, simply change any occurrence of WLSSPI to WBSSPI).

a From the OVO console, select **Application Bank**.

b In the Application Bank window select **WLSSPI → WLSSPI Admin**.

c Double-click **Config WLSSPI**.

d The Introduction window opens.

Click **Next**.

e The configuration editor opens. Add an application server and set the required and conditional properties listed below in Table 1. For more information about using the configuration editor, refer to Appendix B in the WLS SPI or WBS SPI configuration guide.

Table 1 JMX Collector Properties

Property	Description	Level of Configuration	
		Default Properties	Application Server
COLLECT_ METADATA	**Required**. Default: OFF. Set to ON to collect MBean information displayed by the JMB application. The MBean information is used to create UDMs (user defined metrics). Set this property for a subset of your MBean servers (those that have a representative set of MBeans registered). Metadata for each MBean server is temporarily saved to the following file on the managed node (the examples shown are for the WLS SPI; if you have installed the WBS SPI, change any occurrence of wls to wbs): **UNIX**: `/var/opt/OV/wasspi/wls/metadata/<managed_node>/<NAME|ALIAS>.xml`, `/var/opt/OV/metadata/wls/<managed_node>/<NAME|ALIAS>.xml`, `/var/lpp/OV/wasspi/wls/metadata/<managed_node>/<NAME|ALIAS>.xml`, or `/var/lpp/OV/metadata/wls/<managed_node>/<NAME|ALIAS>.xml` **Windows**: *<%OvAgentDir%>\wasspi\wls\metadata\<managed_node>\<NAME|ALIAS>*`.xml` where NAME or ALIAS is the property set for the managed node. When the Gather MBean Data application is run, the MBean information is transferred to the OVO management server and the XML file on the managed node is deleted. The MBean information is saved in a cache file on the OVO management server named `/opt/OV/wasspi/wls/metadata/`*<managed_node>*`/<NAME|ALIAS>.xml`	✓	✓
HOME	**Required**. The directory where the JMX-compliant MBean server is installed. For example, `HOME=C:/jboss-4.0.0`.	✓	✓

Table 1 JMX Collector Properties (cont'd)

Property	Description	Level of Configuration	
		Default Properties	Application Server
JAVA_HOME	**Required**. The directory where Java is installed that is used by the collector. The java engine is expected to be `$JAVA_HOME/bin/java`.	✓	✓
JMB_JAVA_HOME	**Required**. The directory where Java (JDK 1.4.1 or higher) is installed that is used by the JMX Metric Builder on the OVO management server. The JDK must be version 1.4.1 or higher.	✓	✓
JMX_CLASSPATH	**Required**. JMX implementation class path. If you are using JBoss 4.0.0, set this property to: **Windows**: <*jboss_install*>`\lib\jboss-jmx.jar;` <*jboss_install*>`\lib\jboss-common.jar;` <*jboss_install*>`\lib\dom4j.jar` **UNIX**: <*jboss_install*>`/lib/jboss-jmx.jar:` <*jboss_install*>`/lib/jboss-common.jar:`<*jboss_install*>`/lib/dom4j.jar`	✓	✓
NAME	**Required**. The RMI registry name to which the JMX connector binds. When you edited the `JMXRMIConnectorConfig` file, use the same value set for the `com.hp.openview.jmx.connector.bind.name` property.		✓
PORT	**Required**. The port on which the RMI registry server listens. When you edited the `JMXRMIConnectorConfig` file, use the same value set for the `com.hp.openview.jmx.connector.rmi.registry.port` property.		✓
TYPE	**Required**. Set this property to ovrmi if you are not using the MBean server provided by WebLogic or WebSphere application server.	✓	✓
ADDRESS	**Conditional**. Default: localhost. The hostname or IP address where the MBean server is running.		✓
LOGIN	**Conditional**. The login name portion of the credentials used by the JMX connector to perform JAAS authentication.	✓	✓

Table 1 JMX Collector Properties (cont'd)

Property	Description	Level of Configuration	
		Default Properties	Application Server
PASSWORD	**Conditional**. The password portion of the credentials used by the JMX connector to perform JAAS authentication.	✓	✓
RMID_PORT	**Conditional**. The directory where the JMX-compliant MBean server is installed. For example, `HOME=C:/jboss-4.0.0`.	✓	
RMID_START_TIME	**Conditional**. The directory where the JMX-compliant MBean server is installed. For example, `HOME=C:/jboss-4.0.0`.	✓	
VERSION	**Optional**. Default: 1.0. The version number corresponds to the FromVersion/ ToVersion metric elements defined in the metrics definition file. This version number must fall within their range. The FromVersion/ToVersion metric elements are optional.		✓

The following is an example of a JMX collector configuration:

```
JMB_JAVA_HOME = /opt/bea/jdk141
.
.
.
SERVER1_COLLECT_METADATA = ON
SERVER1_HOME = /jboss-4.0.0
SERVER1_JAVA_HOME = /opt/bea/jdk141
SERVER1_JMX_CLASSPATH = /JMX/Sun/lib/jmxri.jar
SERVER1_NAME = myMBeanServer
SERVER1_PORT = 1102
SERVER1_TYPE = ovrmi
SERVER1_LOGIN = testUser
SERVER1_PASSWORD = testPassword
```

When the configuration is saved, the password is encrypted.

f Optionally, click **Save** to save any changes made to the configuration. Once you save your changes, you cannot automatically undo them.

g Click **Finish** or **Next** to save any changes and exit the editor.

If you selected **Next**, the Confirm Operation window opens. Click **OK**.

If you click **Cancel** and made changes to the configuration, those changes remain in the configuration on the management server. To make the changes to the selected managed nodes' configuration, you must select those nodes in the Node Bank window, start the Config WLSSPI application, click **Next** from the Introduction window, click **Next** from the configuration editor, and then click **OK**.

Additional Configuration

After you have configured your MBean server environment, complete the following tasks:

- Task 1: Run the Gather MBean Data Application
- Task 2: Add a UDM Message Group
- Task 3: Assign the Message Group to opc_adm

Task 1: Run the Gather MBean Data Application

To gather the MBean information immediately, run the Gather MBean Data application.

The COLLECT_METADATA property must be set to ON for the managed node on which an MBean server is running (the source server). MBean information is collected from these managed nodes only. Refer to WebLogic/WebSphere MBean Server on page 26 for information on how to set the COLLECT_METADATA property.

To run the Gather MBean Data application, do the following (this example shows the steps using the WLS SPI; if you have installed the WBS SPI, simply change any occurrence of WLSSPI to WBSSPI):

1. At the OVO console, select a node or nodes in the Node Bank window.
2. From the Window menu, select **Application Bank**.
3. In the Application Bank window select **JMX Metric Builder → WLSSPI**.
4. Double-click **Gather MBean Data**.

For more information about this application, see Gather MBean Data on page 88.

Task 2: Add a UDM Message Group

A message group combines management information about similar or related managed objects under a chosen name, and provides status information on a group level. For more information about message groups, refer to the *HP OpenView Operations for UNIX Concepts Guide*.

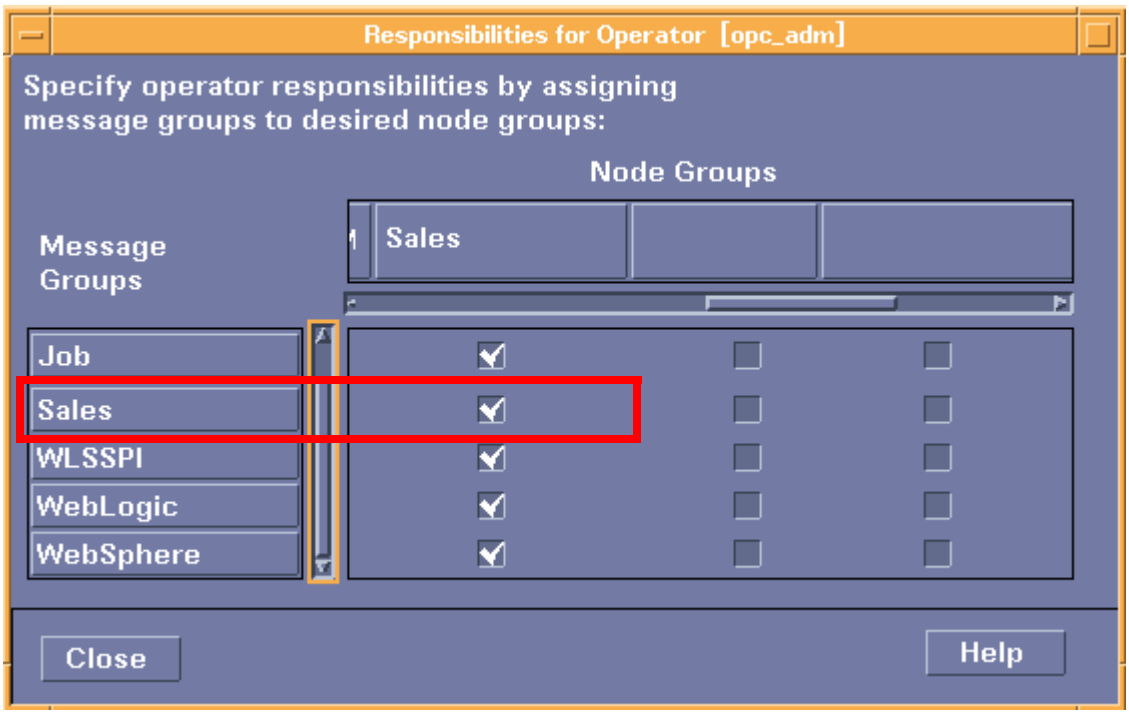
To add a message group, do the following:

1. Open the Add Message Group window.
2. Enter a name (for example, Sales), label, and description.
3. Click **OK**.

Task 3: Assign the Message Group to opc_adm

1. Log on to OVO as the administrator (**opc_adm**).
2. Open the User Bank window, right-click the opc_adm user, and choose **Modify**.
3. In the Modify User:opc_adm user window, click **Responsibilities**.
4. For the Sales Message Group, ensure that all boxes are checked

.



5 Assign the Sales Node or Message Groups to any other appropriate operators.

6 Click **Close**.

4 UDM Development

Once your source and target servers are configured and metadata is being collected from your MBean server(s), you are ready to create and monitor UDMs.

To create and monitor UDMs, do the following:

- Task 1: Update Existing UDMs - If you created UDMs for the WBS SPI or WLS SPI version A.03.10 or earlier, update your UDMs.
- Task 2: Run the JMX Metric Builder - View registered MBeans and create UDMs based on registered MBeans.
- Task 3: Create/Add UDMs Based on PMI Counters - Add UDMs based on PMI counters. UDMs based on PMI Counters cannot be added using the JMB.
- Task 4: Add JMX Actions (Optional) - Add JMX actions to a template or metric. JMX actions cannot be added using the JMB.
- Task 5: Copy Files Generated by the JMB/JMB Plug-in for Eclipse - Copy UDMs and templates generated by the JMB/JMB Plug-in for Eclipse to the OVO management server.
- Task 6: Deploy the UDM File - Deploy the UDM file from the OVO management server to the selected managed node(s).
- Task 7: Create a UDM Template Group and Templates - Create a UDM template group to simplify or customize template distribution, create templates that monitor the UDMs created in task 1, and create templates that define the collection interval (how often the metrics are collected and monitored).
- Task 8: Distribute the Templates - Distribute templates from the OVO management server to the selected managed node(s).
- Task 9: Disable and Re-enable Graphing - If graphing is enabled, disable and re-enable it to create a data source with the new metrics for HP OpenView Performance Manager. If graphing is not enabled, enable it to create a data source with the new metrics for HP OpenView Performance Manager.

Task 1: Update Existing UDMs

If you created UDMs for the WBS SPI or WLS SPI version A.03.10 or earlier, do the following:

1 If your UDM file contains both UDMs based on PMI counters and registered MBeans, separate these metrics into two files (one file containing UDMs based on PMI counters and the other file containing UDMs based on register MBeans). UDMs based on PMI counters cannot be edited using the JMX Metric Builder (JMB).

2 If you want to use the JMB to edit existing UDMs (UDMs based on registered MBeans), update your alarming, graphing, and reporting UDMs to use a metric ID of WBSSPI_1*xxx* or WLSSPI_1*xxx* or JMXUDM_1*xxx* (where *xxx* is a number from 000 through 999).

> Do not update your existing UDMs with metric IDs WBSSPI_07*xx* or WLSSPI_07*xx*. Also, you must not open the UDM file that contains these metrics in the JMB. The JMB converts these metrics to hidden metrics (hidden metrics can only be used to calculate other metrics; they cannot be used as alarming, graphing, nor reporting metrics).

3 If you updated your UDMs, update your templates to reflect the new metric IDs.

4 Move the existing UDM file(s) on the management server so that they are deployed by the Deploy UDM application to the managed nodes.

If you are using the WBS SPI, move the file `/opt/OV/wasspi/wbs/conf/wasspi_wbs_udmDefinitions.xml` (and any other UDM file) to the directory `/opt/OV/wasspi/wbs/conf/workspace/UDMProject`.

If you are using the WLS SPI, move the file `/opt/OV/wasspi/wls/conf/wasspi_wls_udmDefinitions.xml` (and any other UDM file) to the directory `/opt/OV/wasspi/wls/conf/workspace/UDMProject`.

Task 2: Run the JMX Metric Builder

To start the JMB, do the following (this example shows the steps using the WLS SPI; if you have installed the WBS SPI, simply change any occurrence of WLSSPI to WBSSPI):

> If you did not convert your alarming, graphing, and reporting metrics to use metric IDs of WBSSPI_1*xxx* or WLSSPI_1*xxx* or JMXUDM_1*xxx* (where *xxx* is a number from 000 through 999) as described in step 2 of Task 1: Update Existing UDMs on page 42, do not open this UDM file in the JMB.

> Do not open files containing metrics based on PMI counters nor metrics containing JMX actions in the JMB. Currently, the JMB does not support these elements. These elements must be entered manually. For more information, see Task 3: Create/Add UDMs Based on PMI Counters on page 43 and Task 4: Add JMX Actions (Optional) on page 43.

1 At the OVO console, open the Application Bank window.

2 In the Application Bank window select **JMX Metric Builder → WLSSPI**.

3 Double-click **JMX Metric Builder**. Run only one instance of the JMB at a time.

For more information about the JMB, refer to the online help.

Complete the following tasks to create a UDM using the JMB (For more information, refer to the online help):

- Load metadata/MBean information
- Organize MBeans (optional)
- Open the UDM file
- Add a metric

- Change metric visibility (optional)

You may also complete these tasks using the JMB Plug-in for Eclipse. For more information, see The JMX Metric Builder Plug-in for Eclipse on page 13.

Task 3: Create/Add UDMs Based on PMI Counters

If you are creating UDMs based only on PMI counters, you must install and configure the WBS SPI. After you have configured the WBS SPI, no additional configuration is needed to create UDMs based on PMI counters.

You cannot use the JMB to create or edit UDMs based on PMI counters.You must directly edit the UDM file that contain these UDMs. If you open an XML file containing UDMs based on PMI counters in the JMB, all your PMI counter metrics are converted to hidden metrics (hidden metrics can only be used to calculate other metrics; they cannot be used as alarming, graphing, nor reporting metrics).

To create or add UDMs based on PMI counters, do the following:

1. On the OVO management server, edit the `/opt/OV/wasspi/wbs/conf/wasspi_wbs_udmDefinitions.xml`) file or create a new XML file in the `/opt/OV/wasspi/wbs/conf/workspace/UDMProject/` directory.

   UDMs based on PMI counters must be assigned a metric ID in the range of 700 to 799. For more information about the structure and syntax of the UDM file, see Appendix A, Metric Definitions DTD.

2. After you have saved your UDM file, copy the file to the `/opt/OV/wasspi/wbs/conf/workspace/UDMProject/` directory (if it is not already located there). When you deploy your UDMs in Task 6: Deploy the UDM File on page 45, only XML files in this directory are deployed to managed nodes.

Task 4: Add JMX Actions (Optional)

JMX actions are one or more JMX calls (invoke, get, set) performed on an MBean instance or type. Refer to Appendix E, Add JMX Actions, for information about adding JMX actions.

Task 5: Copy Files Generated by the JMB/JMB Plug-in for Eclipse

If both UDMs and templates or either of them were generated using the JMB/JMB Plug-in for Eclipse, they must be copied to the OVO management server. Do the following:

- Task A: Copy UDM Files to the OVO Management Server
- Task B: Copy Template Files to the OVO Management Server

Task A: Copy UDM Files to the OVO Management Server

Copy the UDM files from the managed node/development system to the following directories on the OVO management server:

Files on Managed Node /Development System	**Location on OVO Management Server**
<*User_Defined_Dir*>/<*UDM_File*>.xml	/opt/OV/wasspi/wbs/conf/workspace/UDMProject/ (WebSphere) or /opt/OV/wasspi/wls/conf/workspace/UDMProject/ (WebLogic)
<*UDM_Path*>/UDM/UDM<*project*>.xml (JMB Plug-in for Eclipse)	/opt/OV/wasspi/wls/conf/workspace/UDMProject/ (WebLogic)

where

- <*User_Defined_Dir*> is the directory selected by the user when the UDM file was saved
- <*UDM_File*> is the file name selected by the user when the UDM file was saved
- <*UDM_Path*> is the path displayed in the Preferences window (select **Window** → **Preferences** and select **JMX Metric Builder** in the tree)
- <*project*> is the name of the Eclipse project.

Each UDM file in /opt/OV/wasspi/wbs/conf/workspace/UDMProject/ or /opt/OV/wasspi/wls/conf/workspace/UDMProject/ on the OVO management server must be uniquely named and end with .xml. Verify that the UDMs in each file are uniquely named.

Task B: Copy Template Files to the OVO Management Server

1 Copy the generated template files from the managed node/development system to the following directories on the OVO management server:

Files on Managed Node/Development System	**Location on OVO Management Server**
<*User_Defined_Dir*>/OVOU_Policies/C/set.idx	<*Template_Dir*>/C/
<*User_Defined_Dir*>/OVOU_Policies/C/TEMPLATES/MONITOR/monitor.dat	<*Template_Dir*>/C/TEMPLATES/MONITOR/
<*User_Defined_Dir*>/OVOU_Policies/C/TEMPLATES/TEMPLGROUP/templgroup.dat	<*Template_Dir*>/C/TEMPLATES/TEMPLGROUP/

where

— <*User_Defined_Dir*> is the directory selected by the user when the templates were generated using the JMB/JMB Plug-in for Eclipse

— <*Template_Dir*> is a user-specified directory on the OVO management server. If you are only copying one set of template files (a set of template files consists of the set.idx, monitor.dat, and templgroup.dat files), use the

`/var/opt/OV/share/tmp/OpC_appl/wasspi/udm/wbs_set/` (WebSphere) or `/var/opt/OV/share/tmp/OpC_appl/wasspi/udm/wls_set/` (WebLogic) directory.

If you are copying more than one set of template files to the OVO management server, copy each set of files to a unique <*Template_Dir*> directory.

2 Upload the template information using the `opccfgupld` command. For example, type:

```
/opt/OV/bin/OpC/opccfgupld -verbose -replace \
/var/opt/OV/share/tmp/OpC_appl/wasspi/udm/wbs_set
```

If you copied more than one set of template files to the OVO management server, run this command for each set of templates. For more information about using this command, refer to the *opccfgupld(1M)* man page. For more information about uploading configuration information, refer to the *HP OpenView Operations Developer's Toolkit Application Integration Guide*.

Task 6: Deploy the UDM File

Deploy the UDM file. Running the `Deploy UDM` application creates a single UDM file from all XML files in the `/opt/OV/wasspi/wbs/conf/workspace/UDMProject/` or `/opt/OV/wasspi/wls/conf/workspace/UDMProject/` directory. This single UDM file is deployed to the managed nodes. For more information about this application, see Deploy UDM on page 87. (This example shows the steps using the WLS SPI; if you have installed the WBS SPI, simply change any occurrence of WLSSPI to WBSSPI.)

1 At the OVO console, select a node in the Node Bank window.

2 From the Window menu, select **Application Bank**.

3 In the Application Bank window select **JMX Metric Builder → WLSSPI → Deploy UDM**.

Task 7: Create a UDM Template Group and Templates

Creating a template group for your UDMs allows you to assign multiple templates to a managed node as a single group rather than individually. Templates can be assigned to more than one template group allowing you to customize the templates assigned to managed nodes.

Creating templates allows you to monitor your UDMs and define how often metrics are collected.

To create a template group and templates for your UDMs, do the following:

- Task A: Create a Template Group
- Task B: Create a Metric Template
- Task C: Create a Collector Template

If you modify both the default and sample template groups or either of them along with metric templates, your customizations are overwritten when you upgrade to the next version.

If you copy or create a new template group and templates, these customizations are NOT overwritten when you upgrade to the next version

Task A: Create a Template Group

If you are using the built-in MBean server that comes with the WebLogic/WebSphere application server, the SPIs provide default template groups and templates which you can copy.

To create a new template group, add it from the Message Source Template window (from the OVO console, select the **Window → Message Source Templates**). For more information, refer to the OVO manuals and online help.

Copying an existing template group or creating a new one allows you to keep custom templates separate from the original default templates.

Naming a Template Group

Name a new template group according to how you plan to identify the new monitor and collector templates. For example, you might include UDM in the template group name to clearly indicate that the group is made up of custom templates.

Task B: Create a Metric Template

If you are using the built-in MBean server that comes with the WebLogic/WebSphere application server, the SPIs provide default templates which you can copy.

To create a new template, add it from the Message Source Template window (from the OVO console, select the **Window → Message Source Templates**). For more information, refer to the OVO manuals and online help.

When you create a metric template, you must name it, set conditions, and set threshold monitors.

Naming a Metric Template

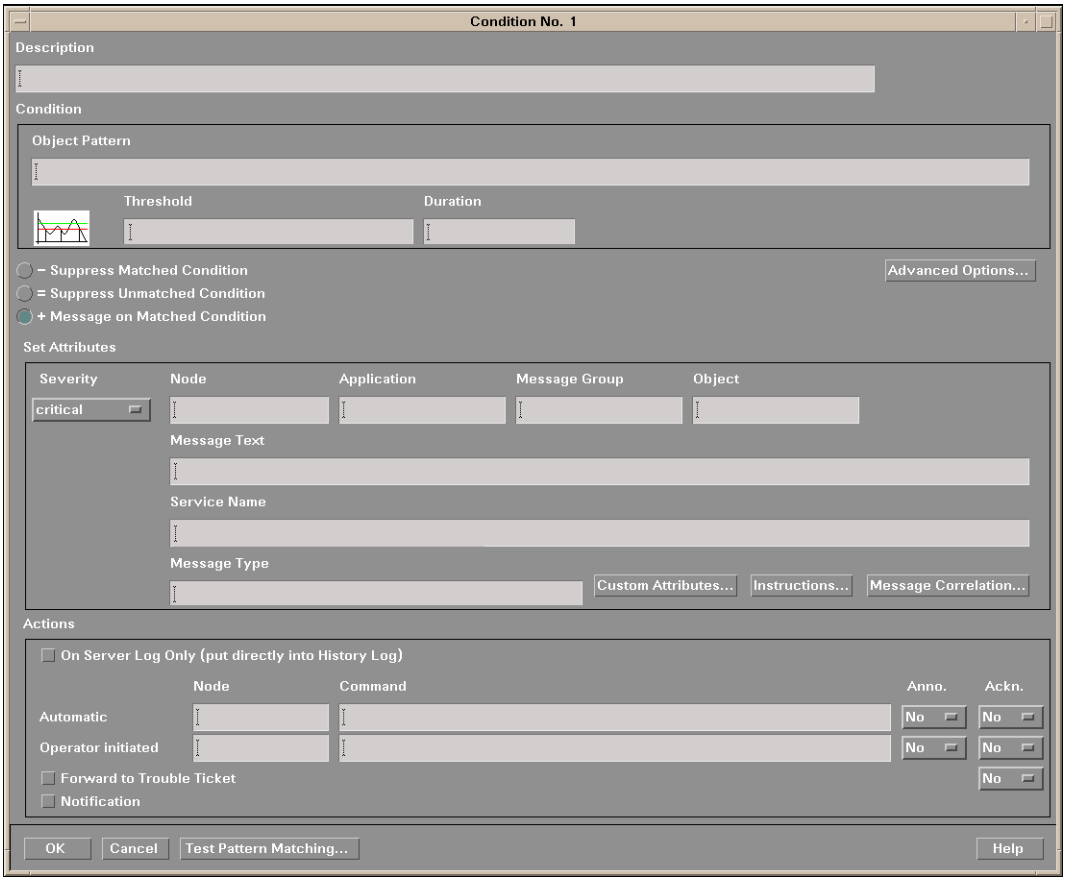
The name you give a metric template *must* match the exposed metric ID of the UDM used in the template. For example, if you are creating a template to use the metric SALES_1001, you must name the template SALES_1001.

Setting Conditions

To set metric conditions, do the following:

1. From the OVO console, select the **Window → Message Source Templates**.
2. In the Message Source Templates window, open your UDM template group.
3. Double-click the desired metric which opens the Message and Suppress Conditions window.

4 Click **Conditions**. The Condition window opens.



Common items to edit are:

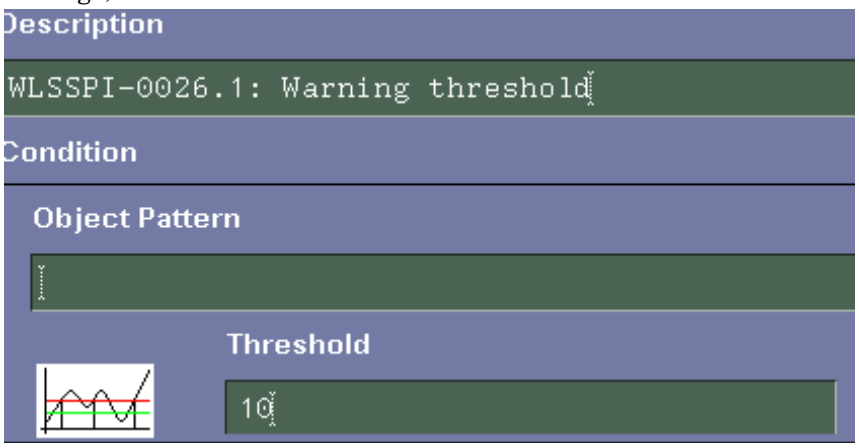
- **Threshold**. Enter a value for the metric data that, when exceeded, would signify a problem either about to occur or already occurring.
- **Duration**. The length of time that the established threshold can be exceeded by the incoming data values for a metric before an alarm is generated.
- **Severity**. The level assigned by the OVO administrator to a message, based on its importance in a given operator's environment. Click **Severity** to select the desired severity setting.
- **Message Group**. The message group to which this message is filtered. Use the message group you configured in Task 2: Add a UDM Message Group on page 38.
- **Message Text**. Structured, readable piece of information about the status of a managed object, an event related to a managed object, or a problem with a managed object. Be careful not to modify any of the parameters—surrounded by <> brackets, beginning with $—in a message.
- **Actions**. Response to a message that is assigned by a message source template or condition. This response can be automatic or operator-initiated. This section provides the ability to generate Performance Manager graphs or reports, or to add custom programs.

  Automatic action. Action triggered by an incoming event or message. No operator intervention is involved. The automatic action delivered with the WLS SPI generates a snapshot report that shows the data values at the time the action was triggered from an exceeded threshold. You can view the report in the message Annotations.

Operator-initiated action. Action used to take corrective or preventive actions in response to a given message. Unlike automatic actions, these actions are triggered only when an operator clicks a button. The operator-initiated action delivered with the WLS SPI allows you to view a graph of the metric whose exceeded threshold generated the message along with other related metric values (Click **Perform Action** within a message's details window).

For more information about this window, refer to the OVO manuals and online help.

The following example of the Condition window shows a threshold setting of 10 for metric WLSSPI-0026. This metric monitors the total number of times per minute clients must wait for an available EJB (enterprise java bean). A value of more than 10 would start to impact the server response time the client experiences, generating an alarm (a warning message).



5 Click **OK**.

6 Distribute the template as described in Task 8: Distribute the Templates on page 51.

Setting Threshold Monitors

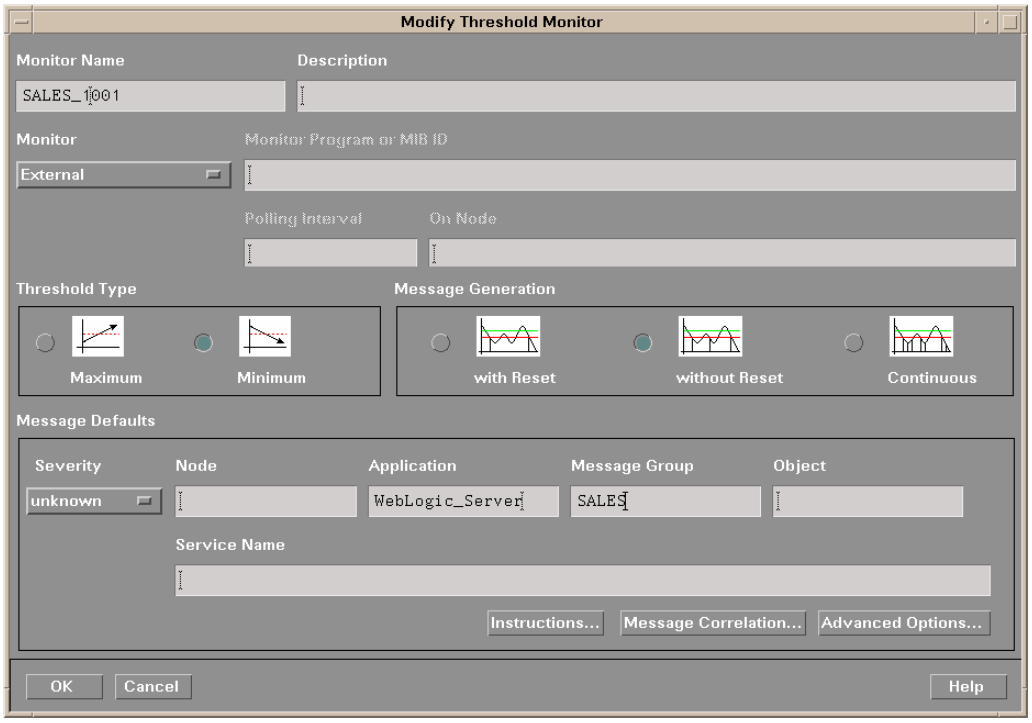
To set threshold monitors, do the following:

1 Open the **Message Source Templates** window.

2 Open the UDM template group.

3 Select a template.

4 Click **Modify**. The Modify Threshold Monitor window opens.



5 Modify the Message Generation:

- **With Reset**: Alarms are generated once when the threshold value is exceeded. At the same time a reset threshold value is activated. Only when the reset threshold value is exceeded, does the original threshold value become active again. Then when the threshold value is again exceeded, another alarm is generated and the process starts all over again.
- **Without Reset**. Alarms are generated once when the monitoring threshold value is exceeded. Alarms reset automatically when metric values are no longer in violation of the thresholds and are generated again when the threshold is exceeded.
- **Continuously**. Messages are sent/alerts generated each time the metric values are collected and the threshold is exceeded.

6 Set the message group to which this message is filtered. Use the message group you configured in Task 2: Add a UDM Message Group on page 38.

7 Click **OK**.

8 Re-distribute the modified templates as described in Task 8: Distribute the Templates on page 51.

Task C: Create a Collector Template

If you are using the built-in MBean server that comes with the WebLogic/WebSphere application server, the SPIs provide default templates which you can copy.

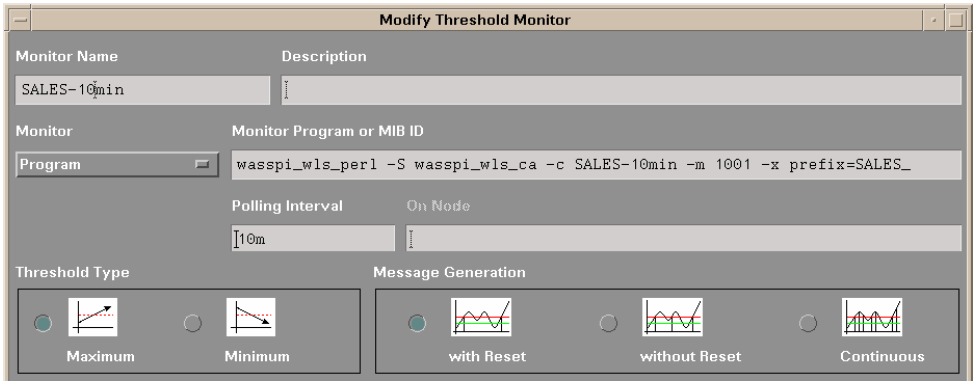
To create a new template, add it from the Message Source Template window (from the OVO console, select the **Window → Message Source Templates**). or more informationRefer to the OVO manuals and online help.

When you create a collector template, you must name it and set threshold monitors.

Naming and Setting Threshold Monitors for a Collector Template

To name and set the threshold monitors of a collector template, do the following:

1. From the OVO console, select the **Window → Message Source Templates**.
2. In the Message Source Templates window, open your UDM template group.
3. Select the collector template to modify.
4. Click **Modify...**. The Modify Threshold Monitor window opens.



5. In the Monitor Name text box, enter the name of the collector template.

   The name you give a copied collector template can be based on the collection (polling) interval of all the metrics to be collected. For example, if you are collecting sales metrics every 10 minutes, you could name the collector template SALES-10m.

   The collector command of this collector template must include the new name.

6. In the Monitor Program or MIB ID text box, enter the collector command (`wasspi_wls_perl -S wasspi_wls_ca` or `wasspi_wbs_perl -S wasspi_wbs_ca`) followed by these options:

Option	Description
`-c`	**Required**. The collector template name (entered in the Monitor Name text box). Example: `-c SALES-10m`
`-m`	The metric number(s) to be collected. Example: `-m 1001`
`-x` *prefix*	The prefix of the UDMs to be collected. This prefix must match the prefix you used in task 1 of this chapter. Example: `-x prefix=SALES_`

   Additional options can be specified for the collector command. Refer to the Using the Collector/Analyzer Command with Parameters section of the SPI Configuration Guide for more details about this command.

7. Edit the Polling Interval.

   For example, enter 10m to specify that the collector template collects UDMs every 10 minutes.

8. Click **OK**.
9. Distribute the template as described in Task 8: Distribute the Templates on page 51.

Syntax Examples

The examples that follow are for the WLS SPI. If you have installed the WBS SPI, replace any occurrence of wls with wbs.

```
wasspi_wls_perl -S wasspi_wls_ca -c SALES-10min -m 1000-1005,1010
-x prefix=SalesUDM_
wasspi_wls_perl -S wasspi_wls_ca -c SALES-15min -m 1100-1120
-x prefix=SalesUDM_
```

Task 8: Distribute the Templates

Deploy templates to the managed nodes:

1. Open the Node Bank window and from the Actions menu select **Agents → Install/Update SW & Config**.
2. In the Target Nodes section select the **Nodes in List Requiring Update** radio button.
3. In the Install/Update Software and Configuration window select the **Templates** check box.
4. Select **Force Update**. Click **OK**. The following message is displayed in the Message Browser:

   ```
   The following configuration information was successfully distributed:
   Templates
   ```

The templates are now distributed to the selected node group. Monitors can now begin running according to their specific collection interval.

Task 9: Disable and Re-enable Graphing

WBS SPI and WLS SPI can be used with HP OpenView Performance Manager to generate graphs showing the collected metric values. To collect the metric values of the UDMs you just created, you must restart the data collection by disabling and enabling graphing.

1. If graphing has been enabled, disable it:
   a. At the OVO console, open the Node Bank window and select a node or groups of nodes on which you want to disable graphing.
   b. Open the Application Bank window.
   c. In the Application Bank window select **JMX Metric Builder → WLSSPI**.
   a. Double-click **UDM Graph Disable**.
2. From the SPI, enable graphing
   a. From the OVO console, open the Node Bank window and select a node or groups of nodes on which you want to enable graphing.
   b. Open the Application Bank window.
   c. In the Application Bank window select **JMX Metric Builder → WLSSPI**.
   d. Double-click **UDM Graph Enable**.

Allow sufficient collection intervals to occur before attempting to view graphs using OpenView Performance Manager (must be purchased seperately).

A Metric Definitions DTD

This Appendix contains:

- Metric Definition DTD
- Sample XML Files
- Explanation of each element in the DTD with the help of examples

The metric definitions DTD provides the structure and syntax for the UDM XML file. The WBS SPI and WLS SPI use this DTD to parse and validate the UDM file. The DTD is described and a sample UDM file is shown in the sections that follow.

The sections that follow assume you are familiar with XML and DTDs.

On a managed node, the metric definitions DTDs are located in the following directory:

Operating System	**Directory**
UNIX	`/var/opt/OV/wasspi/wbs/conf/` or `/var/opt/OV/wasspi/wls/conf/`
UNIX (new non-root HTTPS managed node)	`/var/opt/OV/conf/wbs/` or `/var/opt/OV/conf/wls/`
AIX	`/var/lpp/OV/wasspi/wbs/conf/` or `/var/lpp/OV/wasspi/wls/conf/`
AIX (new non-root HTTPS managed node)	`/var/lpp/OV/conf/wbs/` or `/var/lpp/OV/conf/wls/`
Windows	`%OvAgentDir%\wasspi\wbs\conf\` or `%OvAgentDir%\wasspi\wls\conf\`

Because the DTD files are used at runtime, you should not edit, rename, or move them.

If you are creating UDMs using PMI counters, you must manually edit the UDM file. Otherwise, you edit the UDM file using the JMX Metric Builder (JMB).

The following is a list of elements described in this appendix and the element hierarchy. An element's attribute(s) are enclosed by curly braces ({}) following the element. Required attributes are in **bold**. Sample XML codes are given later to further explain the elements.

```
MetricDefinitions
  Metrics
    Metric+ {id, name, alarm, report, graph, previous, description}
      MBean+ {instanceType, dataType}
        FromVersion? {server, update}
        ToVersion? {server, update}
        ObjectName
        Attribute
        AttributeValueMapping?
          Map+ {from, to}
        AttributeFilter* {type, name, operator, value}
        InstanceID?
          ObjectnameKey
          Attribute
      Calculation+
        FromVersion? {server, update}
        ToVersion? {server, update}
        AggregationKeys
          AggregationKey+
        Formula
      PMICounter+ {instanceType, impact}
        FromVersion? {server, update}
        ToVersion? {server, update}
        Path
        ID
        Load? {data}
        Stat? {data}
      JMXActions? {id}
        JMXAction {id}
         FromVersion? {server, update}
         ToVersion? {server, update}
         JMXCalls+ {id}
           ObjectName
           Set {id}
             Attribute
             Value
               Numeric {type}
                 Formula
               String {value}
               Boolean {value}
           Get {id}
             Attribute
           Invoke+ {id}
             Operation
             Parameters
               Parameter+
                 Numeric {type}
                   Formula
                 String {value}
                 Boolean {value}
```

Sample 1

Metric 10 uses metric mbean1 in its calculation. This calculated metric applies to all WebLogic Server versions. However, the MBean metric on which it is based has changed. Originally the MBean for metric 10 was introduced on server version 6.0, service pack 1. However in version 6.1, the attribute name changed, and this change remains the same up to the current server version 9.2.

```
<Metric id="mbean1" alarm="no">
  <MBean >
    <FromVersion server="6.0" update="1"/>
    <ToVersion server="6.099"/>
    <ObjectName>*:*,Type=ExecuteQueue</ObjectName>
    <Attribute>ServicedRequestTotalCount</Attribute>
  </MBean>
  <MBean >
    <FromVersion server="6.1"/>
    <ObjectName>*:*,Type=ExecuteQueue</ObjectName>
    <Attribute>ServicedRequestCount</Attribute>
  </MBean>
</Metric>
<Metric id="JMXUDM_1010" alarm="yes">
  <Calculation>
    <Formula>
      (delta(mbean1) / interval(mbean1))*1000)
    </Formula>
  </Calculation>
</Metric>
```

In the above example metric mbean1 is used to calculate metric JMXUDM_1010. Mbean1 is a hidden metric (hidden metrics can only be used to calculate other metrics; they cannot be used as alarming, graphing, nor reporting metrics).

If the server version is 6.0 - 6.099, the collector collects data from the attribute `ServicedRequestTotalCount` of object name `*:*,Type=ExecuteQueue`. If the server version is 6.1 and above the collector collects data from the attribute `ServicedRequestCount` of object name `*:*,Type=ExecuteQueue`.

The collector uses this data to calculate the value of metric JMXUDM_1010 using the formula: `(delta(mbean1) / interval(mbean1))*1000)`

Sample 2

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE JMXActions (View Source for full doctype...)>
<JMXActions>
  <JMXAction>
    <JMXCalls>
      <ObjectName>*:*,Type=JMSServerConfig</ObjectName>
      <Set>
        <Attribute>MessagesMaximum</Attribute>
        <Value>
          <Numeric>
            <Formula>JMSServerConfig_MessagesMaximum + (5-5)</Formula>
          </Numeric>
        </Value>
      </Set>
      <Get>
        <Attribute>MessagesMaximum</Attribute>
      </Get>
    </JMXCalls>
  </JMXAction>
  <JMXAction>
    <JMXCalls>
      <ObjectName>*:*,Type=ApplicationConfig</ObjectName>
      <Invoke>
        <Operation>stagingEnabled</Operation>
        <Parameters>
          <Parameter>
            <String value="examplesServer" />
          </Parameter>
        </Parameters>
      </Invoke>
    </JMXCalls>
  </JMXAction>
</JMXActions>
```

The above XML file can be used to modify or obtain the value of an Mbean attribute.

In the first JMX action the collector parses the XML and sets the value of the attribute `MessageMaximum` of the Mbean `*:*,Type=JMSServerConfig` to the numeric value obtained from the formula `JMSServerConfig_MessagesMaximum + (5-5)`.

The value of the attribute `MessageMaximum` is then obtained using the Get element.

In the second JMX action, for the Mbean `*:*,Type=ApplicationConfig`, the operation `stagingEnabled` is invoked using the string value `“examplesServer”`.

Sample Metric Definition Document

In this section a sample metric definition document is given to illustrate how you may create user-defined metrics. The sample document also contains examples of calculated metrics.

WAS SPI collector uses the metric definition file to determine which metrics to collect and their type (MBean, PMI and so on). The metric definition file also helps the collector determine the MBeans to query and the attributes to use.

A sample XML file is included on the management server in `/opt/OV/wasspi/udm/conf/wls/UDMMetrics-sample.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MetricDefinitions SYSTEM "MetricDefinitions.dtd">
<!-- sample UDM metrics configuration File -->

<MetricDefinitions>
  <Metrics>

<!-- The following metrics illustrate some of the options
     available when creating user-defined metrics.
-->

  <!-- The following metric uses an MBean that can have
       multiple instances in the MBean server. Note that
       JMX-compliant pattern-matching can be used in the
       MBean ObjectName tag.
  -->

  <Metric id="WLSSPI_1000" name="UDM_1000" alarm="yes">
    <MBean instanceType="multi">
      <FromVersion server="6.0" update="1"/>
      <ObjectName>*:*,Type=ExecuteQueueRuntime</ObjectName>
      <Attribute>PendingRequestCurrentCount</Attribute>
    </MBean>
  </Metric>

  <!-- The following 2 metrics are "base" metrics.
       They are used in the calculation of a "final"
       metric and are not alarmed, reported, or graphed
       themselves. Base metrics may have an 'id' that
       begins with a letter (case-sensitive) followed by
       any combination of letters, numbers, and underscore.
       Base metrics normally have alarm="no".
  -->

  <Metric id="JVM_HeapFreeCurrent" alarm="no" >
    <MBean instanceType="single">
      <FromVersion server="6.0" update="1"/>
      <ObjectName>*:*,Type=JVMRuntime</ObjectName>
      <Attribute>HeapFreeCurrent</Attribute>
    </MBean>
  </Metric>
  <Metric id="JVM_HeapSizeCurrent" alarm="no">
    <MBean>
      <FromVersion server="6.0" update="1"/>
      <ObjectName>*:*,Type=JVMRuntime</ObjectName>
      <Attribute>HeapSizeCurrent</Attribute>
    </MBean>
  </Metric>

  <!-- The following metric illustrates a calculated metric.
       The calculation is based on the previous 2 "base"
       metrics.
  -->
```

```
  <Metric id="WLSSPI_1005" name="B1005_JVMMemUtilPct"
   alarm="yes" graph="yes">
    <Calculation>
      <FromVersion server="6.0" update="1"/>
      <Formula>((JVM_HeapSizeCurrent-JVM_HeapFreeCurrent)
       /JVM_HeapSize  Current)*100</Formula>
    </Calculation>
  </Metric>

  <!-- The following metric illustrates a mapping from the
       actual string value returned by the MBean attribute
       to a numeric value so that an alarming threshold can
       be specified in a monitor template. Note that the
      'datatype' must be specified as 'string'.
  -->

  <Metric id="WLSSPI_1001" alarm="yes" report="no">
    <MBean dataType="string">
      <ObjectName>*:*,Type=ServerRuntime</ObjectName>
      <Attribute>State</Attribute>
      <AttributeValueMapping>
        <Map from="Running" to="1"/>
        <Map from="Shutdown Pending" to="2"/>
        <Map from="Shutdown In Progress" to="3"/>
        <Map from="Suspended" to="4"/>
        <Map from="Unknown" to="5"/>
      </AttributeValueMapping>
    </MBean>
  </Metric>

  <!-- Metric IDs that are referenced from the collector
       command line must have a prefix followed by
       4 digits. The default prefix is 'JMXUDM_'.
       The 'prefix' option must be used on the command
       line for the following metric since this metric has a
       different prefix other than 'JMXUDM_'.
       Example:
         wasspi_wls_ca -c FIRST_CLIENT_60-5MIN
          -x prefix=Testing_ -m 992 ...
  -->

  <Metric id="Testing_0992" name="Testing_Metric"
   alarm="yes">
    <MBean>
      <ObjectName>*:*,Type=ServerRuntime</ObjectName>
      <Attribute>OpenSocketsCurrentCount</Attribute>
    </MBean>
  </Metric>

  </Metrics>
</MetricDefinitions>
```

AggregationKeys and AggregationKey Elements

The AggregationKeys and AggregationKey elements are used when performing aggregated functions (such as sum and count) on multi-instance metrics. For MBeans, an aggregated key is the JMX ObjectName key or DMS noun type (Oracle AS).

If the AggregationKeys and AggregationKey elements are not specified, the metric value is aggregated at the application server level.

Supporting metric subclasses must implement the corresponding `com.hp.openview.wasspi.metric.AggregateByKeys` interface.

Hierarchy

```
AggregationKeys?
  AggregationKey+
```

The AggregationKeys and AggregationKey elements are children elements of the Calculation element.

The AggregationKeys and AggregationKey elements do not contain any attributes.

Syntax

```
<!ELEMENT AggregationKeys (AggregationKey+)>
<!ELEMENT AggregationKey (#PCDATA)>
```

Example

```
<AggregationKeys>
  <AggregationKey>oc4j_ear</AggregationKey>
  <AggregationKey>SERVLETs</AggregationKey>
</AggregationKeys>
```

Attribute Element

The Attribute element defines the MBean attribute name. Specify this element consistently when defining multi-instance metric calculations.

Hierarchy

```
Attribute
```

The Attribute element is a child element of the Get, InstanceID, MBean, and Set elements.

The Attribute element does not contain any child elements nor attributes.

Syntax

```
<!ELEMENT Attribute (#PCDATA)>
```

Example

```
<Attribute>ServicedRequestCount</Attribute>
```

As explained in Sample 1 on page 55, for version 6.1 and above the collector will collect data about the `ServicedRequestCount` attribute of the MBean.

AttributeFilter Element

The Attribute element provides basic filtering of MBeans based on an MBean attribute.

Hierarchy

```
AttributeFilter* {type, name, operator, value}
```

The AttributeFilter element is a child element of the MBean element.

The AttributeFilter element does not contain any child elements.

Attributes

Attribute	Type/Values	Default Value	Description
type	“include,” “exclude”	“include”	**Optional**. Specifies if an MBean that matches this filter should be included or excluded from consideration by the data collector.
name	text	N/A	**Required**. The MBean attribute on which to apply the filter.
operator	“initialSubString,” “finalSubString,” “anySubString,” “match,” “gt,” “geq,” “lt,” “leq,” “eq,”	N/A	**Required**. The filter to apply. “initialSubString,” “finalSubString,” “anySubString,” and “match” can be used with MBean attributes that return text values. “gt,” “geq,” “lt,” “leq,” “eq” can be used for MBean attributes that return numeric values. For more information about filtering MBeans, refer to the JMX documentation.
value	text or number	N/A	**Required**. The value to compare. The metric definition creator is responsible for making sure the value data type matches the data type of the corresponding MBean attribute.

Syntax

```
<!ELEMENT AttributeFilter EMPTY>
<!ATTLIST AttributeFilter type  (include | exclude) “include”
                          name      CDATA #REQUIRED
                          operator (initialSubString |
                                    finalSubString |
                                    anySubString | match |
```

```
                    gt | geq | lt | leq  | eq)
                    #REQUIRED
          value     CDATA #REQUIRED >
```

Example

```
<AttributeFilter name="MessagesMaximum" operator="lt" value="500"/>
```

In this example, the attribute `MessageMaximum` is filtered out if its value is less than 500. This attribute may be included or excluded from data collection by the collector.

AttributeValueMapping Element

The AttributeValueMapping element specifies numeric values that should be substituted for the values returned by the MBean attribute. Each AttributeValueMapping element contains a number of Map elements. Each Map element specifies one value to be mapped. The Map element can be used to convert string attributes to numbers so they can be compared to a threshold.

Hierarchy

```
AttributeValueMapping?
  Map+ {from, to}
```

The AttributeValueMapping element is a child element of the MBean element.

The AttributeValueMapping element does not contain any attributes

Syntax

```
<!ELEMENT AttributeValueMapping (Map+)>
```

Example

```
<AttributeValueMapping>
  <Map from="Running" to="1"></Map>
  <Map from="Shutdown Pending" to="2"></Map>
  <Map from="Shutdown In Progress" to="3"></Map>
  <Map from="Suspended" to="4"></Map>
  <Map from="Unknown" to="5"></Map>
</AttributeValueMapping>
```

In the above example a string value collected by the collector (“Running”, “Shutdown Pending”) is mapped to an integer (1, 2). This integer value is used by OVO policies to generate alarms/messages. See Sample Metric Definition Document on page 57.

Boolean Element

The Boolean element defines the boolean value used by the operation.

Hierarchy

```
Boolean {value}
```

The Boolean element is a child element of the Parameter and Value elements.

The Boolean element does not contain any child elements.

Attribute

Attribute	Type/ Values	Default Value	Description
value	“true,” “false”	N/A	**Required**. The boolean value used by the operation.

Syntax

```
<!ELEMENT Boolean    EMPTY>
<!ATTLIST Boolean    value    (true | false) #REQUIRED
```

Example

```
<Boolean value=”true”/>
```

Calculation Element

The Calculation element is used when the data source of the metric is a calculation using other defined metrics. The Calculation element contains a Formula element whose content is a string that specifies the mathematical manipulation of other metric values to obtain the final metric value. The metrics are referred to in the calculation expression by their metric ID. The collector can perform calculations that combine one or more metrics to define a new metric. The result of the calculation is the metric value.

Hierarchy

```
Calculation+
  FromVersion? {server, update}
  ToVersion? {server, update}
  AggregationKeys?
    AggregationKey+
  Formula
```

The Calculation element is a child element of the Metric element.

The Calculation element does not contain any attributes.

Syntax

```
<!ELEMENT Calculation (FromVersion?, ToVersion?, AggregationKeys?, Formula)>
```

Example

```
<Calculation>
  <Formula>
    (delta(mbean1) / interval(mbean1))*1000)
  </Formula>
</Calculation>
```

In the above example the collector calculates the value of a metric (See Sample 1 on page 55) using the formula `(delta(mbean1) / interval(mbean1))*1000)`.

Formula Element

The Formula element's content is a string that specifies the mathematical manipulation of other metric values to obtain the final metric value. The metrics are referred to in the formula by their metric ID. The collector calculates formulas that combine one or more metrics to define a new metric. The result of the formula is the metric value.

Hierarchy

```
Formula
```

The Formula element is a child element of the Calculation and Numeric elements.

The Formula element does not contain any child elements nor attributes.

Syntax

```
<!ELEMENT Formula (#PCDATA)>
```

A formula must use syntax as follows.

- Operators supported are +, -, /, *, and unary minus.
- Operator precedence and associativity follow the Java model.
- Parentheses can be used to override the default operator precedence.
- Allowable operands are metric IDs and literal doubles.

A metric ID can refer to either an MBean metric, another calculated metric, or a PMICounter metric. Literal doubles can be specified with or without the decimal notation. The metric ID refers to the id attribute of the Metric element in the metric definitions document.

Functions

The formula parser also supports the following functions. All function names are lowercase and take a single parameter which must be a metric ID.

- **delta** returns the result of subtracting the previous value of the metric from the current value.
- **interval** returns the time in milliseconds that has elapsed since the last time the metric was collected.
- **sum** returns the summation of the values of all the instances of a multi-instance metric.
- **count** returns the number of instances of a multi-instance metric.
- **prev** returns the previous value of the metric.

Examples

The following example defines a metric whose value is the ratio (expressed as a percent) of Metric_1 to Metric_3.

```
<Formula>(Metric_1 / Metric_3) *100</Formula>
```

The following example can be used to define a mbean that is a rate (number of times per second) for mbean1. See Sample 1 on page 55.

```
<Formula>
  (delta(mbean1) / interval(mbean1))*1000)
</Formula>
```

FromVersion and ToVersion Elements

The FromVersion and ToVersion elements are used to specify the versions of the application server for which the data source element is valid.

The following algorithm is used for determining what application server version is supported by each metric source element within the Metric element.

1. If a FromVersion element is not present, no lower limit exists to the server versions supported by this metric.
2. If a FromVersion element is present, the server attribute indicates the lowest server version supported by this metric. If an update attribute exists, it additionally qualifies the lowest server version supported by specifying the lowest service pack or patch supported for that version.

3 If a ToVersion element is not present, no upper limit exists to the server versions supported by this metric.

4 If a ToVersion tag is present, the server attribute indicates the highest server version supported by this metric. If an update attribute exists, it additionally qualifies the server version supported by specifying the highest service pack or patch supported for that version.

Hierarchy

```
FromVersion? {server, update}
ToVersion? {server, update}
```

The FromVersion and ToVersion elements are child elements of the Calculation, JMXAction, MBean, and PMICounter elements.

The FromVersion and ToVersion elements do not contain any child elements.

Attributes

Attribute	Type/ Values	Default Value	Description
server	numeric string	N/A	**Required**. The primary server version.
update	numeric string	“*”	Optional. The secondary server version, such as “1” for service pack 1. A “*” indicates that no secondary version is specified.

Syntax

```
<!ELEMENT FromVersion (EMPTY)>
<!ELEMENT ToVersion (EMPTY)>

<!ATTLIST FromVersion   server CDATA #REQUIRED
                        update CDATA “*” >

<!ATTLIST ToVersion     server CDATA #REQUIRED
                        update CDATA “*” >
```

Example

```
<FromVersion server="6.0" update="1"/>
<ToVersion server="6.099"/>
```

As explained in Sample 1 on page 55, the collector will collect data for server versions 6.0 to 6.099.

Get Element

The Get element returns the value of the specified attribute.

Hierarchy

```
Get {id}
  Attribute
```

The Get element is a child element of the JMXCalls element.

Attribute

The JMXCalls element attribute is described in the following table.

Attribute	Type/ Values	Default Value	Description
id	ID	N/A	Optional. A unique identifier for this element.

Syntax

```
<!ELEMENT Get (Attribute)>
<!ATTLIST Get id       ID       #IMPLIED>
```

Example

```
<Get>
  <Attribute>MessagesMaximum</Attribute>
</Get>
```

In this example the collector obtains the value of the attribute `MessagesMaximum`. See Sample 2 on page 56.

ID Element

The ID element is the PMI data id to be retrieved from the counter. This information is located in the following file:

WebSphere server version 6.1: *<websphereHome>*`/plugins/com.ibm.ws.runtime_6.1.0.jar`
WebSphere server version 6.0: *<wbs6-home>*`/lib/pmi.jar`
WebSphere server version 5.1: *<wbs5.1-home>*`/lib/pmi.jar`
WebSphere server version 5: *<wbs5-home>*`/lib/pmi.jar`
WebSphere server version 4: *<wbs4-home>*`/lib/perf.jar`

Hierarchy

```
ID
```

The ID element is a child element of the PMICounter element.

The ID element does not contain any child elements nor attributes.

Syntax

```
<!ELEMENT ID (#PCDATA)>
```

Example

```
<ID>3</ID>
```

This example shows that the collector will retrieve PMI data id from counter 3.

InstanceId Element

The InstanceId element is the unique identifier of a multi-instance MBean.

Hierarchy

```
InstanceID?
  ObjectnameKey
  Attribute
```

The InstanceId element is a child element of the MBean element.

The InstanceId element does not contain any attributes.

Syntax

```
<!ELEMENT InstanceId (ObjectNameKey | Attribute)>
```

Example

```
<InstanceId>*:*,Type=JMSServerConfig</InstanceId>
```

Invoke Element

The Invoke executes an MBean operation with the given parameters.

Hierarchy

```
Invoke+ {id}
  Operation
  Parameters
    Parameter
      Numeric {type}
        Formula
      String {value}
      Boolean
{value}
```

The Invoke element is a child element of the JMXCalls element.

Attribute

Attribute	Type/ Values	Default Value	Description
id	ID	N/A	Optional. A unique identifier for this element.

Syntax

```
<!ELEMENT Invoke (Operation, Parameters?)>
<!ATTLIST Invoke id       ID       #IMPLIED>
```

Example

```
<Invoke>
  <Operation>stagingEnabled</Operation>
    <Parameters>
      <Parameter>
        <String value="examplesServer" />
      </Parameter>
    </Parameters>
</Invoke>
```

In the above example the MBean operation `stagingEnabled` is invoked by passing the string parameter `examplesServer`. See Sample 2 on page 56.

JMXAction Element

The JMXAction element contains one or more JMXCalls elements and all are executed in the order defined. A JMXAction can optionally be associated with specific versions of the application server using the FromVersion and ToVersion elements.

Hierarchy

```
JMXAction {id}
  FromVersion? {server, update}
  ToVersion? {server, update}
  JMXCalls+ {id}
    ObjectName
    Set {id}
      Attribute
      Value
        Numeric {type}
          Formula
        String {value}
        Boolean {value}
    Get {id}
      Attribute
    Invoke+ {id}
      Operation
      Parameters
        Parameter
          Numeric {type}
            Formula
          String {value}
          Boolean {value}
```

The JMXAction element is a child element of the JMXActions element.

Attribute

Attribute	Type/ Values	Default Value	Description
id	ID	N/A	Optional. A unique identifier for this element.

Syntax

```
<!ELEMENT JMXAction (FromVersion?, ToVersion?, JMXCalls+)>
<!ATTLIST JMXAction id       ID       #IMPLIED>
```

Example

```
<JMXAction>
  <JMXCalls>
    <ObjectName>*:*,Type=ApplicationConfig</ObjectName>
    <Invoke>
      <Operation>stagingEnabled</Operation>
      <Parameters>
        <Parameter>
          <String value="examplesServer" />
        </Parameter>
      </Parameters>
    </Invoke>
  </JMXCalls>
</JMXAction>
```

The above example indicates that one JMX call will be performed on an MBean. See Sample 2 on page 56.

JMXActions Element

The JMXActions element contains one or more JMXAction elements. All elements matching the server version are executed.

Hierarchy

```
JMXActions? {id}
  JMXAction {id}
    FromVersion? {server, update}
    ToVersion? {server, update}
    JMXCalls+ {id}
      ObjectName
      Set {id}
        Attribute
        Value
          Numeric {type}
            Formula
          String {value}
          Boolean {value}
      Get {id}
        Attribute
        Value
      Invoke+ {id}
        Operation
        Parameters
          Parameter
            Numeric {type}
              Formula
            String {value}
            Boolean {value}
```

The JMXActions element is a child element of the Metric element.

Attribute

Attribute	Type/ Values	Default Value	Description
id	ID	N/A	Optional. A unique identifier for this element.

Syntax

```
<!ELEMENT JMXActions (JMXAction+)>
<!ATTLIST JMXActions id       ID       #IMPLIED>
```

Example

```
<JMXAction>
  <JMXCalls>
    <ObjectName>*:*,Type=JMSServerConfig</ObjectName>
    <Get>
      <Attribute>MessagesMaximum</Attribute>
```

```
    </Get>
  </JMXCalls>
</JMXAction>
```

See Sample 2 on page 56 for a detailed example.

JMXCalls Element

The JMXCalls element contains one or more JMX calls (invoke, get, or set) that operate on a specific MBean or type of MBean. The MBean instance or type is specified by the ObjectName element.

Hierarchy

```
JMXCalls+ {id}
  ObjectName
  Set {id}
    Attribute
    Value
      Numeric {type}
        Formula
      String {value}
      Boolean {value}
  Get {id}
    Attribute
    Value
  Invoke+ {id}
    Operation
    Parameters
      Parameter
        Numeric {type}
          Formula
        String {value}
        Boolean
{value}
```

The JMXCalls element is a child element of the JMXAction element.

Attribute

Attribute	Type/ Values	Default Value	Description
id	ID	N/A	Optional. A unique identifier for this element.

Syntax

```
<!ELEMENT JMXCalls (ObjectName, (Set | Get | Invoke)+)>
<!ATTLIST JMXCalls id       ID       #IMPLIED>
```

Example

```
<JMXCalls>
  <ObjectName>*:*,Type=ApplicationConfig</ObjectName>
  <Invoke>
    <Operation>stagingEnabled</Operation>
    <Parameters>
      <Parameter>
        <String value="examplesServer" />
      </Parameter>
    </Parameters>
  </Invoke>
</JMXCalls>
```

In the above example, for MBean `*:*, Type=JMSServerConfig`, the operation `stagingEnabled` is invoked by passing the string parameter `examplesServer`. See Sample 2 on page 56

Load Element

The Load element is the type of data to be retrieved and thus various time-based values are available for retrieval. Use the data attribute to specify which value to retrieve.

Hierarchy

`Load? {`**`data`**`}`

The Load element is a child element of the PMICounter element.

The Load element does not contain any child elements.

Attributes

Attribute	Type/ Values	Default Value	Description
data	“mean,” “sum,” “weight,” “current”	N/A	**Required**. The time-based data to retrieve.

Collector can collect the mean/sum/weight/current value of a PMI counter using the value specified for the Load element.

Syntax

```
<!ELEMENT Load EMPTY>
<!ATTLIST Load data (mean | weight | sum | current) #REQUIRED>
```

Example

```
<Load data="weight"/>
```

The example above indicates that the "weight" of the PMI counter will be collected.

Map Element

The Map element specifies one value to be mapped in the AttributeValueMapping element. This element can be used to convert string attributes to numbers so they can be compared to a threshold.

Hierarchy

```
Map+ {from, to}
```

The Map element is a child element of the AttributeValueMapping element.

The Map element does not contain any child elements.

Attributes

Attribute	Type/ Values	Default Value	Description
from	text	N/A	**Required**. The value that is to be mapped.
to	text	N/A	**Required**. The new metric value to be returned in place of the mapped value.

Syntax

```
<!ELEMENT Map EMPTY>
<!ATTLIST Map from  CDATA  #REQUIRED
              to    CDATA  #REQUIRED >
```

Example

```
<Map from="Running" to="1"></Map>
```

The above example indicates that the string value "Running" has been mapped to the integer "1". This integer value will be used by the OVO policies to generate messages/ alarms.

MBean Element

The MBean element is used when the data source of the metric is an attribute of a JMX MBean.

Hierarchy

```
MBean+ {instanceType, dataType}
  FromVersion? {server, update}
  ToVersion? {server, update}
  ObjectName
  Attribute
  AttributeValueMapping?
    Map+ {from, to}
  AttributeFilter* {type, name, operator, value}
  InstanceID?
    ObjectnameKey
    Attribute
```

The MBean element is a child element of the Metric element.

Attributes

Attribute	Type/ Values	Default Value	Description
instanceType	“single,” “multi”	“single”	Optional. Indicates if there could be multiple instances of this MBean.
dataType	“numeric,” “parsedNumeric,” “string,” “boolean”	“numeric”	Optional. Indicates if the value returned from the MBean attribute is a numeric, parsed numeric, string, or a boolean value. The parsed numeric value is the java.lang.String parsed into java.lang.Double.

Syntax

```
<!ELEMENT MBean (FromVersion?, ToVersion?, InstanceId?,
                 ObjectName, Attribute,
                  AttributeValueMapping?, AttributeFilter*)>

<!ATTLIST MBean instanceType  (single | multi) “single”
                dataType      (numeric | parsedNumeric | string | boolean)
                              “numeric” >
```

Example

```
<MBean instanceType="single">
  <FromVersion server="6.0" update="1"/>
  <ObjectName>*:*,Type=JVMRuntime</ObjectName>
  <Attribute>HeapFreeCurrent</Attribute>
</MBean>
```

The above example indicates that the collector collects metric data about the attribute `HeapFreeCurrent` of the Mbean `*:*,Type=JVMRuntime`. This data is collected only if the server version is 6.0 or above. Also, see Sample Metric Definition Document on page 57.

MetricDefinitions Element

The MetricDefinitions element is the top-level element within the document. It contains one collection of metrics, consisting of one or more metric definitions.

Hierarchy

```
MetricDefinitions
  Metrics
    Metric+ {id, name, alarm, report, graph, previous, description}
      MBean+ {instanceType, dataType}
      Calculation+
      PMICounter+ {instanceType, impact}
      JMXActions? {id}
```

Syntax

```
<!ELEMENT MetricDefinitions (Metrics)>
```

Metrics and Metric Elements

The Metric element represents one metric. Each metric has a unique ID (for example, “WLSSPI_1001”). If a user-defined metric is an alarming, graphing, or reporting metric, the metric ID must be “prefix<*xxxx*>” where prefix is made up of 3-15 letters (case-sensitive), digits, or underscores (“_”), and <*xxxx*> must be a number from 1000 through 1999. If a user-defined metric is based on a PMI counter, the metric ID must be “prefix<*xxxx*>” where prefix is made up of 3-15 letters (case-sensitive), digits, or underscores (“_”), and <*xxxx*> must be a number from 700 through 799.Otherwise, if the metric is used only within the calculation of another metric, the metric ID must begin with a letter (case-sensitive) and can be followed by any combination of letters, numbers, and underscores (for example, “mbean1”).

A Metric element contains one or more metric source elements that represent the metric data source. Data sources supported are: MBeans, calculations, and PMI counters. Each metric source element is scanned for a FromVersion or ToVersion child element to determine which metric source element to use for the version of the application server being monitored.

Hierarchy

```
Metrics
  Metric+ {id, name, alarm, report, graph, previous, description}
    MBean+ {instanceType, dataType}
    Calculation+
    PMICounter+ {instanceType, impact}
    JMXActions? {id}
```

The Metrics and Metric elements are child elements of the MetricDefinitions element.

Attributes

Attribute	Type/ Values	Default Value	Description
id	ID	N/A	**Required**. The metric ID.
name	text	“”	Optional. The metric name, used for graphing and reporting. The name can be up to 20 characters in length.
alarm	“yes,” “no”	“no”	Optional. If yes, the metric value is sent to the agent through opcmon.
report	“yes,” “no”	“no”	Optional. If yes, the metric value is logged for reporting.
previous	“yes,” “no”	“yes”	Optional. If yes, the metric value is saved in a history file so that deltas can be calculated. If you are not calculating deltas on a metric, set this to "no" for better performance.
graph	“yes,” “no”	“no”	Optional. If yes, the user-defined metric is graphed.
description	text	“”	Optional. A description of the metric.

Syntax

```
<!ELEMENT Metrics (Metric+)>
<!ATTLIST Metrics %reportNameSpace;>
<!ELEMENT Metric ((MBean+| Calculation+| PMICounter+),JMXActions?)>
<!ATTLIST Metric id           ID         #REQUIRED
                 name         CDATA        “”
                 alarm       (yes | no)    “no”
                 report      (yes | no)    “no”
                 graph       (yes | no)    “no”
                 previous    (yes | no)    “yes”
                 description   CDATA       #IMPLIED >
```

Numeric Element

The Numeric element defines the value type and value either passed as a parameter or assigned to an MBean attribute. The Numeric element contains a formula, defined by the Formula element, (for more information, see Formula Element on page 63) that specifies the mathematical manipulation of other metric values. The result of the formula is the value.

Hierarchy

```
Numeric {type}
  Formula
```

The Numeric element is a child element of the Parameter and Value elements.

Attribute

Attribute	Type/ Values	Default Value	Description
type	“short,” “int,” “long,” “double,” “float,” “java.lang.Short,” “java.lang.Integer,” “java.lang.Long,” “java.lang.Double,” “java.lang.Float”	N/A	Optional. The type of numeric parameter used by the operation.

Syntax

```
<!ELEMENT Numeric    (Formula)>
<!ATTLIST Numeric    type    (short | int | long | double |
                              float | java.lang.Short |
                              java.lang.Integer |
                              java.lang.Long |
                              java.lang.Double |
                              java.lang.Float)     #IMPLIED
```

Example

```
<Numeric>
  <Formula>JMSServerConfig_MessagesMaximum + (5-5)</Formula>
</Numeric>
```

The above example indicates that the value obtained from the formula, `JMSServerConfig_MessagesMaximum + (5-5)` will be an integer. See Sample 2 on page 56.

ObjectName Element

The ObjectName element is the JMX-compliant object name of the MBean. The object name can include JMX-compliant pattern matching.

Hierarchy

```
ObjectName
```

The ObjectName element is a child element of the JMXCalls and MBean elements.

The ObjectName element does not contain any child elements nor attributes.

Syntax

```
<!ELEMENT ObjectName (#PCDATA)>
```

Example

```
<ObjectName>*:*,Type=ExecuteQueue</ObjectName>
```

The above example indicates that `*:*,Type=ExecuteQueue` is the JMX-compliant object name of the MBean. See Sample 1 on page 55.

ObjectNameKey Element

The ObjectNameKey uniquely identifies multi-instance MBeans. Specify this element consistently when defining multi-instance metric calculations.

Hierarchy

```
ObjectnameKey
```

The ObjectNameKey element is a child element of the InstanceId element.

The ObjectNameKey element does not contain any child elements nor attributes.

Syntax

```
<!ELEMENT ObjectNameKey (#PCDATA)>
```

Example

```
<ObjectNameKey>Type=JMSServerConfig</ObjectNameKey>
```

The above example identifies multi-instance MBeans of type `JMSServerConfig`.

Operation Element

The Operation element defines the MBean operation to be performed on an attribute.

Hierarchy

```
Operation
```

The Operation element is a child element of the Invoke element.

The Operation element does not contain any child elements nor attributes.

Syntax

```
<!ELEMENT Operation  (#PCDATA)>
```

Example

```
<Operation>stagingEnabled</Operation>
```

This example indicates that the collector must perform the `StagingEnabled` operation on an attribute. See Sample 2 on page 56.

Parameters and Parameter Elements

The Parameters and Parameter elements define the MBean operation parameter values. Parameters must be specified for operations that accept parameters.

Hierarchy

```
Parameters
  Parameter+
    Numeric {type}
      Formula
    String {value}
    Boolean
{value}
```

The Parameters and Parameter elements are child elements of the Invoke element.

The Parameters and Parmaeter elements do not contain any attributes

Syntax

```
<!ELEMENT Parameters (Parameter)+>
<!ELEMENT Parameter  (Numeric | String | Boolean)>
```

Example

```
<Parameters>
  <Parameter>
    <String value="examplesServer"/>
  </Parameter>
</Parameters>
```

The above example indicates that a string parameter “examplesServer” is passed for an operation. See Sample 2 on page 56.

Path Element

The Path element is the location of the counter in the PMI data hierarchy. The content of this element begins with the PMI module name and may contain the wildcard character to specify multiple instances.

Hierarchy

`Path`

The Path element is a child element of the PMICounter element.

The Path element does not contain any child elements nor attributes.

Syntax

```
<!ELEMENT Path (#PCDATA)>
```

Example

```
<Path>threadPoolModule/*</Path>
```

The above example indicates that the collector will obtain the value of a PMI counter using the path `threadPoolModule/*`.

PMICounter Element

The PMICounter element is used when the metric data source is a PMI counter. UDMs based on PMI counters need to be assigned a metric ID in the range of 700 to 799.

Hierarchy

```
PMICounter+ {instanceType, impact}
  FromVersion? {server, update}
  ToVersion? {server, update}
  Path
  ID
  Load? {data}
  Stat? {data}
```

The PMICounter element is a child element of the Metric element.

Attributes

Attribute	Type/ Values	Default Value	Description
instanceType	“single,” “multi”	“single”	Optional. Indicates if there are multiple instances of this counter.
impact	“low,” “medium,” “high,” “maximum”	N/A	**Required**. How the metric affects application server performance.

Syntax

```
<!ELEMENT PMICounter (FromVersion?, ToVersion?, Path, ID,
                      (Load | Stat)?)>
<!ATTLIST PMICounter instanceType (single | multi) "single"
           impact  (low | medium | high | maximum) #REQUIRED>
```

Example

```
<PMICounter instanceType-”multi” impact=”high”>
  <Path>threadPoolModule/*</Path>
  <ID>3</ID>
  <Load data=”weight”/>
</PMICounter>
```

The above example indicates that the collector will collect a PMI counter with ID 3.

Set Element

The Set element assigns a value to the specified attribute.

Hierarchy

```
Set {id}
  Attribute
  Value
    Numeric {type}
      Formula
    String {value}
    Boolean {value}
```

The Set element is a child element of the JMXCalls element.

Attribute

Attribute	Type/ Values	Default Value	Description
id	ID	N/A	Optional. A unique identifier for this element.

Syntax

```
<!ELEMENT Set     (Attribute, Value)>
<!ATTLIST Set     id       ID       #IMPLIED>
```

Example

```
<Set>
  <Attribute>MessagesMaximum</Attribute>
  <Value>
    <Numeric>
      <Formula>JMSServerConfig_MessagesMaximum + (5-5)</Formula>
    </Numeric>
  </Value>
</Set>
```

The above example indicates that the collector will perform JMX Actions on the attribute MessagesMaximum of an Mbean (not mentioned in the example). The collector will then set the value of the attribute MessagesMaximum to the value obtained by the formula `MSServerConfig_MessagesMaximum + (5-5)`. See Sample 2 on page 56.

Stat Element

The Stat element is the type of data to be retrieved and thus various sample-based values are available for retrieval. Use the data attribute to specify which value to retrieve.

Hierarchy

```
Stat? {data}
```

The Stat element is a child element of the PMICounter element.

The Stat element does not contain any child elements.

Attributes

Attribute	Type/Values	Default Value	Description
data	“mean,” “count,” “sumOfSquares,” “variance,” “standardDeviation,” “confidence,”	N/A	**Required**. The sample-based data to retrieve.

Syntax

```
<!ELEMENT Stat EMPTY>
<!ATTLIST Stat data (mean | count | sumOfSquares | variance |
                     standardDeviation | confidence) #REQUIRED>
```

Example

```
<Stat data="mean"/>
```

The code in the above example indicates that the collector will collect the ‘mean’ value from a PMI counter.

String Element

The String element defines the string used by the operation.

Hierarchy

```
String {value}
```

The String element is a child element of the Parameter and Value elements.

The String element does not contain any child elements.

Attribute

Attribute	Type/ Values	Default Value	Description
value	text	N/A	**Required**. The string used by the operation.

Syntax

```
<!ELEMENT String     EMPTY>
<!ATTLIST String     value    CDATA #REQUIRED>
```

Example

```
<String value="examplesServer"/>
```

The above example indicates that a string value `examplesServer` is used by an operation.
Sample 2 on page 56

ToVersion Element

See FromVersion and ToVersion Elements on page 64 for information about the ToVersion element.

Value Element

The Value element is the value to assign to the attribute. The value can be a number, string, or boolean.

Hierarchy

```
Value
  Numeric {type}
    Formula
  String {value}
  Boolean {value}
```

The Value element is a child element of the Set element.

The Value element does not contain any attributes.

Syntax

```
<!ELEMENT Value   (Numeric | String | Boolean)>
```

Example

```
<Value>
  <Numeric>
    <Formula>JMSServerConfig_MessagesMaximum + (5-5)</Formula>
  </Numeric>
</Value>
```

This example indicates that the collector will assign the numeric value obtained from the formula `JMSServerConfig_MessagesMaximum + (5-5)`to an MBean. See Sample 2 on page 56.

B Applications

The WASSPI-UDM-Bldr software bundle installs the following applications:

- Deploy UDM
- Gather MBean Data
- JMX Metric Builder
- UDM Graph Enable/Disable

Along with the applications installed with the WASSPI-UDM-Bldr software bundle, the following WBSSPI/WLSSPI Admin applications can be run even if you are not managing a WebSphere or WebLogic application server (WBSSPI/WLSSPI Admin applications not listed here cannot be run successfully):

- Config WBSSPI/WLSSPI
- Self-Healing Info
- Start/Stop Monitoring
- Start/Stop Tracing
- Verify
- View Error File
- View Graphs

The examples in this appendix are for the WLS SPI. If you have installed the WBS SPI, replace any occurrence of WLSSPI with WBSSPI and wls with wbs.

JMX Metric Builder Application Group

The following applications are found in the WBSSPI or WLSSPI application group found in the JMX Metric Builder application group. These applications require the “root” user permission, therefore it is recommended that this group be assigned to the OVO administrator.

Deploy UDM

Deploys the UDM file from the management server to the selected managed node(s).

Purpose

Deploys the UDM file to the selected managed node(s). UDMs allow you to define your own metrics and monitor applications registered with the WebLogic MBean server.

Function

`Deploy UDM` deploys the UDM file from the management server to the *%OVAgentDir%*`/wasspi/wbs/conf/wasspi_wbs_udmDefinitions.xml`, *%OVAgentDir%*`/conf/wbs/wasspi_wbs_udmDefinitions.xml`, *%OVAgentDir%*`/wasspi/wls/conf/wasspi_wls_udmDefinitions.xml`, or *%OVAgentDir%*`/conf/wls/wasspi_wls_udmDefinitions.xml` file on the selected managed nodes.

All XML files in the `/opt/OV/wasspi/wbs/conf/workspace/UDMProject or /opt/OV/wasspi/wls/conf/workspace/UDMProject` directory are combined to form a single UDM file.

If the UDM file on the management server does not exist or is empty, the following error message appears:

```
The UDM file <filename> does not exist.
```

How to Run

1. At the OVO console, select a node in the Node Bank window.
2. From the Window menu, select **Application Bank**.
3. In the Application Bank window select **JMX Metric Builder** → **WLSSPI** → **Deploy UDM**.

Gather MBean Data

Gathers MBean information from the selected managed node(s).

Required Setup

The COLLECT_METADATA property must be set to ON for the managed node on which an MBean server is running. Gather MBean Data only collects MBean information from these managed nodes.

Purpose

Gathers MBean information from all managed nodes whose COLLECT_METADATA property is set to ON. This information is saved in a cache on the OVO management server.

The MBean information is displayed by the JMX Metric Builder (JMB) application so that you can create UDMs.

Function

Gather MBean Data collects MBean information and saves it to a cache on the OVO management server.

Initially, the MBean information is saved in an XML file on the managed node in `/var/opt/OV/wasspi/wbs/tmp/<NAME | ALIAS>.xml`, `/var/opt/OV/tmp/wbs/<NAME | ALIAS>.xml`, `/var/opt/OV/wasspi/wls/tmp/<NAME | ALIAS>.xml`, or `/var/opt/OV/tmp/wls/<NAME | ALIAS>.xml` where NAME and ALIAS are the properties set for the managed node and ALIAS is always used if it is set.

Once Gather MBean Data has collected the MBean information for a managed node, the MBean information is transferred to the OVO management server and is saved in a cache file named `/opt/OV/wasspi/wbs/metadata/<managed_node>/<NAME | ALIAS>.xml`, or `/opt/OV/wasspi/wls/metadata/<managed_node>/<NAME | ALIAS>.xml`. The XML file on the managed node is deleted.

If a cache file on the OVO management server is no longer collected, it is automatically deleted.

How to Run

1. At the OVO console, select a node or nodes in the Node Bank window.
2. From the Window menu, select **Application Bank**.
3. In the Application Bank window select **JMX Metric Builder → WLSSPI → Gather MBean Data**.

JMX Metric Builder

Launches the JMB. Run only one instance of the JMB at a time.

Required Setup

Complete the following tasks before running the JMB:

- Register your custom MBeans. For more information, see Register Your Custom MBeans on page 25.
- Configure you MBean server environment. For more information, see MBean Server Environment Configuration on page 26.
- Run the Gather MBean Data application. For more information, see Task 1: Run the Gather MBean Data Application on page 38.

Purpose

Launches the JMB allowing you to edit the UDM file and browse MBeans on an MBean server.

Function

JMX Metric Builder allows you do the following:

- Load metadata
- Organize MBeans
- Add a metric
- Change metric visibility
- Remove a metric

UDMs for all OVO managed nodes are maintained in UDM files on the OVO management server. Use the `Deploy UDM` application to distribute the UDM files from the management server to the managed node(s).

How to Run

1. At the OVO console, open the Application Bank window.
2. In the Application Bank window select **JMX Metric Builder** → **WLSSPI** → **JMX Metric Builder**.

UDM Graph Enable/Disable

Starts/stops data collection for UDM graphs.

Purpose

Starts/stops data collection for UDM graphs. Also starts/stops the OVO subagent.

If you have configured UDMs, you can collect data that can be used by HP OpenView Performance Manager.

Function

UDM Graph Enable starts UDM data collection for graphing.

UDM Graph Disable stops UDM data collection for graphing.

How to Run

1. At the OVO console, select a node in the Node Bank window.
2. From the Window menu, select **Application Bank**.
3. In the Application Bank window select **JMX Metric Builder** → **WLSSPI** → **UDM Graph Enable** or **UDM Graph Disable**.

Enable JMB Tracing

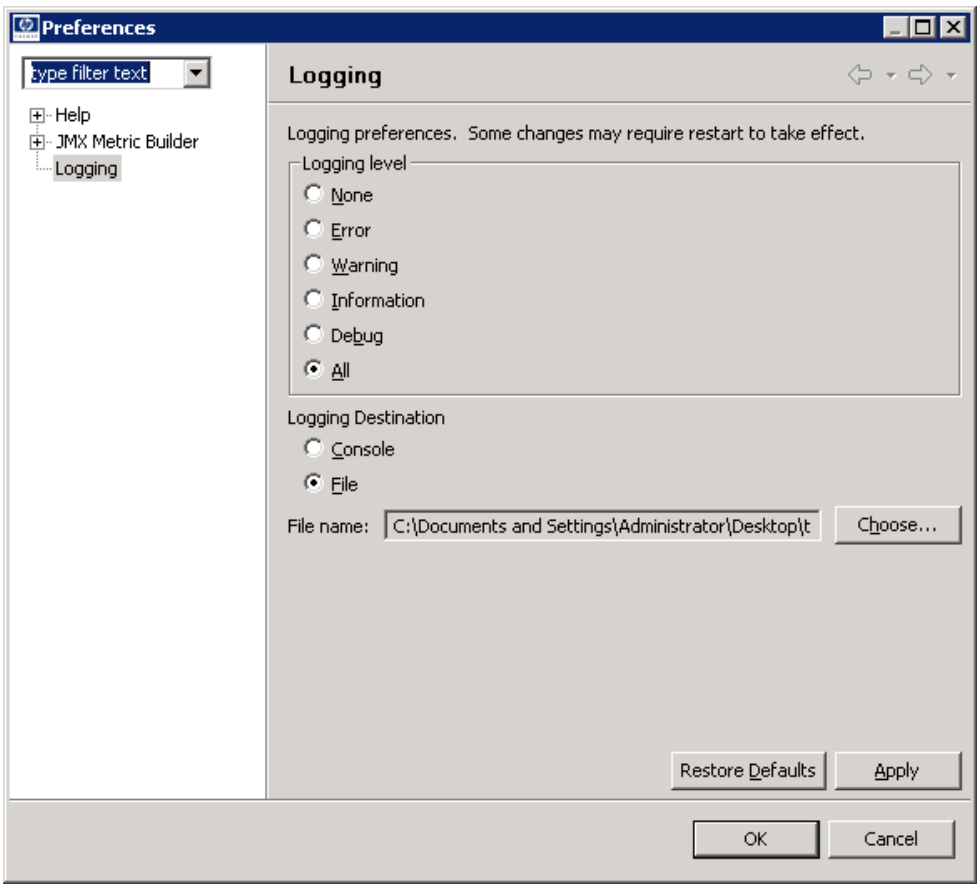
In the earlier versions of JMB, tracing was enabled by launching the Trace JMB application. The Trace JMB application is no longer available. You can now enable JMB tracing through the JMB GUI.

To enable JMB tracing follow these steps:

1. From the JMB GUI, select Window →Preferences. The Preferences window opens.
2. From the Logging pane select a Logging level and the Logging Destination.

   Select Console as the logging destination if you want to view the tracing results in the JMB console window.

   Select File as the logging destination if you want to save the tracing results in a file. Click Choose, to select the file where you want the data to be logged.



3 Click Apply.

4 Click OK.

C JMX Connector Management Interface

The JMX connector exposes the following attributes and methods as its management interface:

Method	Description
`public void start();`	Binds the JMX connector to the RMI registry.
`public void stop();`	Unbinds the JMX connector from the RMI registry.
`public String getHost();`	The hostname where the JMX connector is running.
`public int getRmiRegistryServerPort();`	The port on which the RMI registry server accepts connections.
`public String getProtocol();`	The RMI protocol used (JRMP).
`public String getState();`	Current state of the JMX connector (RUNNING or STOPPED).
`public boolean isRunning();`	Availability of the JMX connector (True if running, false if not running).
`public String getServiceName();`	The RMI bind name used to locate the JMX connector.
`public String getVersion();`	The JMX connector version.
`public long getStartTime();`	The registration time, in milliseconds, of the connector MBean since January 1, 1970, GMT.
`public String getStartTimePretty();`	Start time formatted according to the default locale.
`public String getConfigFileName();`	The configuration file used to configure the JMX connector.

D Authenticating with JBoss UsersRolesLoginModule

To use UsersRolesLoginModule supplied with JBoss when implementing a JMXAuthenticator (for JBoss 4.0 security considerations), do the following:

1. Add the following security configuration to the file *<current-server-config>*`/conf/login-config.xml`:

```
<application-policy name = "OVJMXConnectorSecurity">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
flag = "required" />
  </authentication>
</application-policy>
```

2. Assign the security configuration name to a property in the `properties-service.xml` file. The property is retrieved in the JMXAuthenticator implementation:

```
<attribute name="Properties">
  login.modules.config.name=OVJMXConnectorSecurity
</attribute>
```

3. Add UsersRolesLoginModule files, `users.properties` and `roles.properties`, to the JMX Connector SAR root. See the JBoss documentation for a description of these files.

4. Implement `com.hp.openview.wasspi.connector.rmi.JMXAuthenticator` to make use of the new security configuration. The following is an example implementation:

```
package JAAS;

import com.hp.openview.wasspi.connector.rmi.JMXAuthenticator;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import org.jboss.security.auth.callback.UsernamePasswordHandler;

/**
 * Interface to convert remote credentials to a JAAS Subject.
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2001</p>
 * <p>Company: </p>
 * @author not attributable
 * @version 1.0
 */
public class JMXAuthenticatorImpl implements JMXAuthenticator {

  /**
   * Authenticates the RMI client with the supplied credentials.
```

```
   * @param credentials a 2-element string array containing the login and
   * password used to authenticate the user.
   * @return the authenticated Subject
   */
   public Subject authenticate(Object credentials) {

     // Get login and password.
     Object [] arrayObject = (Object[])credentials;
     String [] loginPassword = new String [] {(String)arrayObject[0],
(String)arrayObject[1]};

     if (loginPassword[0] == null)
       throw new SecurityException("User name not specified.");

     if (loginPassword[1] == null)
       throw new SecurityException("Password not specified.");


     Subject resultSubject = null;
     try
     {
       // Set username and password in the handler.
       UsernamePasswordHandler handler
         = new UsernamePasswordHandler(loginPassword[0],
loginPassword[1]);

       // This property is defined in jboss-service.xml.  It is set to the
       // JMX Connector login configuration defined in login-config.xml.
       String loginConfigName =
System.getProperty("login.modules.config.name", "");

       // LoginContext will instantiate a new Subject
       LoginContext lc = new LoginContext(loginConfigName, handler);

       // authenticate the Subject
       lc.login();

       // get the authenticated Subject
       resultSubject = lc.getSubject();

       if (resultSubject == null) {
         throw new SecurityException("Null Subject.");
       }
     }
     catch (LoginException le)
     {
       SecurityException se = new SecurityException("Authentication
failed.");
       se.initCause(le);

       throw se;
     }
     return resultSubject;
   }
}
```

5 Deploy the JMXAuthenticator implementation class file to the JMX Connector SAR root. The following is a recursive Windows directory listing of the JMX Connector SAR root after deployment:

```
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\JAAS
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\JMXRMIConnectorConfig
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\JMX_RMI_Connector.jar
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\META-INF
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\roles.properties
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\users.properties
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\JAAS\JMXAuthenticatorImpl.class
C:\jmx\jboss-4.0.0\server\default\deploy\OVJMXConnector.sar\META-INF\jboss-service.xml
```

6 Add the JMXAuthenticator implementation class name to the `JMXRMIConnectorConfig` file:

```
com.hp.openview.jmx.connector.authenticator=JAAS.JMXAuthenticatorImpl
```

7 When configuring the SPI, set LOGIN and PASSWORD to one of the entries in `users.properties`. For example, entry **`"admin=adminpass"`** translates to LOGIN=**`admin`** and PASSWORD=**`adminpass`**.

E Add JMX Actions

JMX actions are one or more JMX calls (invoke, get, set) performed on one or more MBean instances.

JMX actions are executed from the collector command. A single JMX call can be defined on the command itself or multiple calls can be defined in an XML file (such as a UDM file).

This appendix contains the following information:

- Using the Collector Command Parameters - a description of the collector command parameters and how to define a single JMX call using the collector command
- Defining JMX Actions in XML - how to define and implement JMX actions in an XML file
- Defining JMX Actions in a Metric Definition - how to define and implement JMX actions in a UDM file

Using the Collector Command Parameters

To implement a JMX action using the collector command, include the `-a` parameter and then choose the `-mbean`, `-xml`, or `-m` parameter.

`-mbean` performs a single JMX call specified in the command line:

```
-a -mbean <objectname>
   { -get <attribute> |
     -invoke <operation> [[-type <parameter_type>] <parameter_value>]... |
     -set <attribute> <value>
   } [-i <servers>] [-o <object>]
```

`-xml` performs one or more JMX calls defined in the specified XML file:

```
-xml <filename> [-i <servers>] [-o <object>]
```

`-m` performs one or more JMX calls defined in the UDM file for the specified metric:

```
-m <metric_id> [-i <servers>] [-o <object>]
```

The following are the JMX actions parameters that can be used in the collector command:

Parameter	Description
-a **Required**	(action) Indicates a JMX action is performed. **Syntax**: `-a`
-i	(include) Allows you to list specific servers on which to perform the JMX action(s). If this parameter is not specified, the JMX action(s) are performed on all configured servers. **Syntax**: `-i <server_name>` **Example**: `-i server1,server3`
-m	(metric) Specifies the metric ID containing the JMX action(s) to perform. This metric ID must be defined in a UDM file. This option may not be used with the -mbean nor -xml options. **Syntax**: `-m <metric_id>` **Example**: `-m TestUDM_1000`

<table>
<tr><th>Parameter</th><th colspan="2">Description</th></tr>
<tr><td>-mbean</td><td colspan="2">Performs a JMX call on the specified MBean(s). This option may not be used with the -m nor -xml options.

Syntax: -mbean <objectname> <action>

Example:
-mbean WebSphere:type=ThreadPool,* -set growable true

where <action> (a JMX call) is one of the following:</td></tr>
<tr><td></td><td>-get</td><td>Returns the value of the specified attribute.
Syntax: -mbean <objectname> -get <attribute>

Example: -get maximumSize</td></tr>
<tr><td></td><td>-invoke [-type]</td><td>Executes an MBean operation with the specified parameters. -type is optional and can be used to specify a parameter type. -type enables support for operation overloading.

Syntax: -mbean <objectname> -invoke <operation> [[-type <parameter_type>] <parameter_value>]...

where <parameter_type> is one of the following: short, int, long, double, float, boolean, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Double, java.lang.Float, java.lang.Boolean, and java.lang.String.

Example: -invoke setInstrumentationLevel -type java.lang.String pmi=L -type boolean true</td></tr>
<tr><td></td><td>-set</td><td>Assigns the specified value to the specified attribute.

Syntax: -mbean <objectname> -set <attribute> <value>

Example: -set growable true</td></tr>
<tr><td>-o</td><td colspan="2">(object) Specifies an MBean instance.

Syntax: -o <mbean_instance>

Example: -o exampleJMSServer</td></tr>
<tr><td>-xml</td><td colspan="2">Specifies the XML file that contains the JMX action(s) to perform (include the fully-qualified path). This option may not be used with the -m nor -mbean options.

Syntax: -xml <filename>

Example: -xml /tmp/myJMXActions.xml</td></tr>
</table>

WBS SPI Command Line Examples

The following are examples of performing a single JMX call from the collector command line:

- Set the maximum size for an alarming thread pool to 500 (where <$OPTION(instancename)> specifies an alarming instance):

```
wasspi_wbs_perl -S wasspi_wbs_ca -a -mbean WebSphere:type=ThreadPool,*
-set maximumSize 500 -o <$OPTION(instancename)>
```

- Set the instrumentation levels to low on all PMI modules:

```
wasspi_wbs_perl -S wasspi_wbs_ca -a -mbean WebSphere:type=Perf,*
-invoke setInstrumentationLevel -type java.lang.String pmi=L
```

- Set the ThreadPool maximumSize attribute to 50 on multiple MBean instances:

```
wasspi_wbs_perl -S wasspi_wbs_ca -a -mbean WebSphere:type=ThreadPool,*
-set maximumSize 50 -i server1
```

- Set the ThreadPool maximumSize attribute to 50 on a specific MBean instance:

```
wasspi_wbs_perl -S wasspi_wbs_ca -a -mbean WebSphere:type=ThreadPool,*
-set maximumSize 50 -i server1 -o MessageListenerThreadPool
```

- Invoke an operation on a specific MBean instance:

```
wasspi_wbs_perl -S wasspi_wbs_ca -a -mbean WebSphere:type=Perf,*
-invoke setInstrumentationLevel pmi=m true -i server1 -o PerfMBean
```

- Get the ThreadPool maximumSize attribute:

```
wasspi_wbs_perl -S wasspi_wbs_ca -a -mbean WebSphere:type=ThreadPool,*
-get maximumSize -i server1
```

WLS SPI Command Line Examples

The following are examples of performing a single JMX call from the collector command line:

- Set the maximum threads for an alarming WebLogic execute queue to 50 (where <$OPTION(instancename)> specifies an alarming instance):

```
wasspi_wls_perl -S wasspi_wls_ca -a
-mbean “PetStore:*,Type=ExecuteQueueConfig”
-set ThreadsMaximum 50 -o <$OPTION(instancename)>
```

- Set the MessagesMaximum attribute to 25000 on multiple MBean instances:

```
wasspi_wls_perl -S wasspi_wls_ca -a -mbean *:*,Type=JMSServerConfig
-set MessagesMaximum 250000 -i examplesServer
```

- Set the MessagesMaximum attribute to 25000 on a specific MBean instance:

```
wasspi_wls_perl -S wasspi_wls_ca -a -mbean *:*,Type=JMSServerConfig
-set MessagesMaximum 250000 -i examplesServer -o examplesJMSServer
```

- Invoke an operation on multiple MBean instances:

```
wasspi_wls_perl -S wasspi_wls_ca -a -mbean *:*,Type=ApplicationConfig
-invoke staged -i examplesServer
```

- Get the MessagesMaximum attribute:

```
wasspi_wls_perl -S wasspi_wls_ca -a -mbean *:*,Type=JMSServerConfig
-get MessagesMaximum -i examplesServer
```

Defining JMX Actions in XML

To implement JMX actions defined in an XML file, on the collector command line, include the `-a` and `-xml` parameters and specify the XML file to use. The JMX actions defined in the specified XML file are performed.

1. Create an XML file containing JMX actions. Follow the syntax for the JMXActions element defined by the metric definitions DTD (see Appendix A, Metric Definitions DTD for more details about each element and attribute and XML File Examples on page 103 for example XML files).
2. Copy and rename a collector template (refer to the previous section for more detailed information about completing this step).
3. Modify the command line and remove the -m parameter and its specified metric numbers.
4. Modify the command line and include the -a and -xml parameters followed by the name of the XML file. Include the fully-qualified path with the filename.
5. Distribute the new template (refer to the previous section for more detailed information about completing this step).

XML File Examples

The following is an example XML file for WBS SPI (available online in `/var/opt/OV/wasspi/wbs/conf/wbs_JMXActions-sample.xml` or `/var/opt/OV/conf/wbs/wbs_JMXActions-sample.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE JMXActions SYSTEM "JMXActions.dtd">

<!-- @WHAT_STRING@ -->

<!-- Sample JMX Actions XML -->

<JMXActions>
  <!-- The Following action modifies maximum size
       and sets growable to true on all thread pool instances.
       'Get' elements are included only for validation.
  -->
 <JMXAction>
  <JMXCalls>
    <ObjectName>WebSphere:type=ThreadPool,*</ObjectName>
    <Set>
      <Attribute>maximumSize</Attribute>
      <!-- Do a non-destructive set for demo only.
           ThreadPool_maximumSize is defined in UDM file wbs_UDMMetrics-sample
           Therefore, UDM configuration needs to specify wbs_UDMMetrics-sample.
      -->
      <Value>
        <Numeric>
          <Formula>ThreadPool_maximumSize + (2-2)</Formula>
        </Numeric>
      </Value>
    </Set>
    <!-- Optional Get to validate prior Set. -->
    <Get>
      <Attribute>maximumSize</Attribute>
    </Get>
    <Set>
      <Attribute>growable</Attribute>
```

```
      <Value>
        <Boolean value="true"/>
      </Value>
    </Set>
    <!-- Optional Get to validate prior Set. -->
    <Get>
      <Attribute>growable</Attribute>
    </Get>
  </JMXCalls>
 </JMXAction>

  <!-- The Following action will recursively set
       instrumentation levels to low on all PMI modules. The
       getInstrumentationLevelString operation is defined only for validation.
  -->

 <JMXAction>
  <JMXCalls>
    <ObjectName>WebSphere:type=Perf,*</ObjectName>
    <Invoke>
      <Operation>setInstrumentationLevel</Operation>
      <Parameters>
        <Parameter>
          <String value="pmi=l"/>
        </Parameter>
        <Parameter>
          <Boolean value="true"/>
        </Parameter>
      </Parameters>
    </Invoke>
    <!-- Optional to validate prior setInstrumentationLevel. -->
    <Invoke>
      <Operation>getInstrumentationLevelString</Operation>
    </Invoke>
  </JMXCalls>
 </JMXAction>
</JMXActions>
```

The following is an example XML file for WLS SPI (available online in `/var/opt/OV/wasspi/wls/conf/wls_JMXActions-sample.xml` or `/var/opt/OV/conf/wls/wls_JMXActions-sample.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE JMXActions SYSTEM "JMXActions.dtd">

<!-- @WHAT_STRING@ -->

<!-- Sample JMX Actions XML -->

<JMXActions>
  <!-- This action will modify maximum
       messages on all JMS server instances.
       A 'Get' element is defined only for validation.
  -->
 <JMXAction>
  <JMXCalls>
    <ObjectName>*:*,Type=JMSServerConfig</ObjectName>
    <!-- Rewrite same value.
         JMSServerConfig_MessagesMaximum is defined in UDM file
         wls_UDMMetrics-sample Therefore, UDM configuration needs to specify
         wls_UDMMetrics-sample.
    -->
    <Set>
      <Attribute>MessagesMaximum</Attribute>
      <Value>
```

```
        <Numeric>
          <Formula>JMSServerConfig_MessagesMaximum + (5-5)</Formula>
        </Numeric>
      </Value>
    </Set>
    <Get>
      <Attribute>MessagesMaximum</Attribute>
    </Get>
  </JMXCalls>
 </JMXAction>
  <!-- The following action demonstrates an operation invoke.
  -->
 <JMXAction>
  <JMXCalls>
    <ObjectName>*:*,Type=ApplicationConfig</ObjectName>
    <!-- A non-modifying operation for demonstration only. -->
    <Invoke>
      <Operation>stagingEnabled</Operation>
      <Parameters>
        <Parameter>
          <String value="examplesServer"/>
        </Parameter>
      </Parameters>
    </Invoke>
  </JMXCalls>
 </JMXAction>
</JMXActions>
```

Command Line Examples

The following are examples of implementing a JMX action from the collector command line using the example JMX actions XML file:

- `wasspi_wbs_perl -S wasspi_wbs-ca -a -xml /var/opt/OV/wasspi/wbs/conf/wbs_JMXActions-sample.xml -i examplesServer`
- `wasspi_wls_perl -S wasspi_wls-ca -a -xml /var/opt/OV/wasspi/wls/conf/wls_JMXActions-sample.xml -i examplesServer`

Defining JMX Actions in a Metric Definition

To implement JMX actions defined in a UDM file, on the collector command line, include the -a and -m parameters and specify the metric ID containing the action. The JMX actions defined for the specified metric are performed.

1. Edit the UDM file containing the metric that will perform JMX actions. You cannot create JMX actions using the JMB. Instead, you must manually edit the UDM file. Follow the syntax for the JMXActions element defined by the metric definitions DTD (see Appendix A, Metric Definitions DTD for more details about each element and attribute and UDM File Examples on page 106 for example UDM files).
2. Copy and rename a collector template (refer to the previous section for more detailed information about completing this step).
3. Modify the command line and remove the -m parameter and its specified metric numbers.
4. Modify the command line and include the -a and -m parameter followed by the metric ID.
5. Distribute the new template (refer to the previous section for more detailed information about completing this step).

UDM File Examples

The following are example metrics for WBS SPI (available online in the `/opt/OV/wasspi/udm/conf/wbs/wbs_UDMMetrics-sample.xml` file):

```
<!-- The Following metric defines a JMX action which will modify maximum size
     and set growable to true on all thread pool instances.  'Get' elements
     are included only for validation.
-->
<Metric id="TestUDM_1000" description="systemModule.freeMemory" alarm="yes">
  <PMICounter instanceType="single" impact="low">
    <FromVersion server="5.0"/>
    <Path>systemModule</Path>
    <ID>3</ID>
  </PMICounter>
  <JMXActions>
   <JMXAction>
    <JMXCalls>
      <ObjectName>WebSphere:type=ThreadPool,*</ObjectName>
      <Set>
        <Attribute>maximumSize</Attribute>
        <!-- Do a non-destructive set for demo only. -->
        <Value>
          <Numeric>
            <Formula>ThreadPool_maximumSize + (2-2)</Formula>
          </Numeric>
        </Value>
      </Set>
      <!-- Optional Get to validate prior Set. -->
      <Get>
        <Attribute>maximumSize</Attribute>
      </Get>
      <Set>
        <Attribute>growable</Attribute>
        <Value>
          <Boolean value="true"/>
        </Value>
      </Set>
      <!-- Optional Get to validate prior Set. -->
      <Get>
```

```
        <Attribute>growable</Attribute>
       </Get>
     </JMXCalls>
    </JMXAction>
  </JMXActions>
</Metric>

<!-- The Following metric defines a JMX action wich will recursively set
     instrumentation levels to low on all PMI modules.  The
     getInstrumentationLevelString operation is defined only for validation.
-->
<Metric id="TestUDM_1001" description="systemModule.cpuUtilization" alarm="yes">
  <PMICounter instanceType="single" impact="low">
    <FromVersion server="5.0"/>
    <Path>systemModule</Path>
    <ID>1</ID>
  </PMICounter>
  <JMXActions>
   <JMXAction>
    <JMXCalls>
      <ObjectName>WebSphere:type=Perf,*</ObjectName>
      <Invoke>
        <Operation>setInstrumentationLevel</Operation>
        <Parameters>
         <Parameter>
           <String value="pmi=l"/>
         </Parameter>
         <Parameter>
           <Boolean value="true"/>
         </Parameter>
        </Parameters>
      </Invoke>
      <!-- Optional to validate prior setInstrumentationLevel. -->
      <Invoke>
        <Operation>getInstrumentationLevelString</Operation>
      </Invoke>
    </JMXCalls>
   </JMXAction>
  </JMXActions>
</Metric>
```

The following are example metrics for WLS SPI (available online in the `/opt/OV/wasspi/udm/conf/wls/wls_UDMMetrics-sample.xml` file):

```
<!-- The Following metric defines a JMX action wich will modify maximum
     messages on all JMS server instances.
     A 'Get' element is defined only for validation.
-->
<Metric id="TestUDM_1000" alarm="yes">
  <MBean instanceType="multi">
    <ObjectName>*:*,Type=JMSServerRuntime</ObjectName>
    <Attribute>MessagesCurrentCount</Attribute>
  </MBean>
  <JMXActions>
   <JMXAction>
    <JMXCalls>
      <ObjectName>*:*,Type=JMSServerConfig</ObjectName>
      <!-- Rewrite same value. -->
      <Set>
        <Attribute>MessagesMaximum</Attribute>
        <Value>
          <Numeric>
            <Formula>JMSServerConfig_MessagesMaximum + (5-5)</Formula>
          </Numeric>
        </Value>
      </Set>
      <Get>
```

```
        <Attribute>MessagesMaximum</Attribute>
      </Get>
    </JMXCalls>
   </JMXAction>
  </JMXActions>
</Metric>

<!-- The Following metric defines a JMX action which demonstrates an operation
     invoke.
-->
<Metric id="TestUDM_1001" alarm="yes">
  <MBean instanceType="multi">
    <ObjectName>*:*,Type=ApplicationConfig</ObjectName>
    <Attribute>LoadOrder</Attribute>
  </MBean>
  <JMXActions>
   <JMXAction>
    <JMXCalls>
      <ObjectName>*:*,Type=ApplicationConfig</ObjectName>
      <!-- A non-modifying operation for demonstration only. -->
      <Invoke>
        <Operation>stagingEnabled</Operation>
        <Parameters>
         <Parameter>
           <String value="examplesServer"/>
         </Parameter>
        </Parameters>
      </Invoke>
    </JMXCalls>
   </JMXAction>
  </JMXActions>
</Metric>
```

Command Line Examples

The following are examples of implementing a JMX action from the collector command line using the example metrics:

- Use the sample UDM TestUDM_1000 in the `wbs_UDMMetrics-sample.xml` file:

  ```
  wasspi_wbs_perl -S wasspi_wbs-ca -a -m TestUDM_1000 -i examplesServer
  ```

- Use the sample UDM TestUDM_1001 in the `wls_UDMMetrics-sample.xml` file:

  ```
  wasspi_wls_perl -S wasspi_wls-ca -a -m TestUDM_1001 -i examplesServer
  ```

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X