HP Operations Smart Plug-in for User Defined Metrics

for HP Operations Manager for HP-UX, Linux, and Solaris

Software Version: 7.04

Installation and Configuration Guide

Document Release Date: December 2010
Software Release Date: December 2010

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/ Oracle AS (version 10gR3 only) 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 Notice

Trademark Notices

UNIX® is a registered trademark of The Open Group.

Microsoft® and Windows® are US registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Documentation Updates

The title page of this document 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://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

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

Visit the HP Software Support Online web site at:

www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web 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 register for an HP Passport user ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

1 Introduction to HP Smart Plug-in for User Defined Metrics

The Smart Plug-in for User Defined Metrics (SPI JMB/UDM) enables you to collect data from the following supported Application servers:

- Oracle WebLogic Application Server (WebLogic SPI)
- IBM WebSphere Application Server (WebSphere SPI)
- Oracle Application Server (Oracle AS SPI) (version 10gR3 only)

For information on the supported application servers versions, see the Support Matrix (SUMA) link:

http://support.openview.hp.com/selfsolve/document/KM323488

About the User Defined Metric SPI

You can create User Defined Metrics (UDMs) to gather data from application MBeans registered in the supported Application servers. You can create UDMs by using the JMX Metric Builder (JMB) or JMB Plug-in for Eclipse by editing the metrics definition XML file (also referred to as the UDM file).

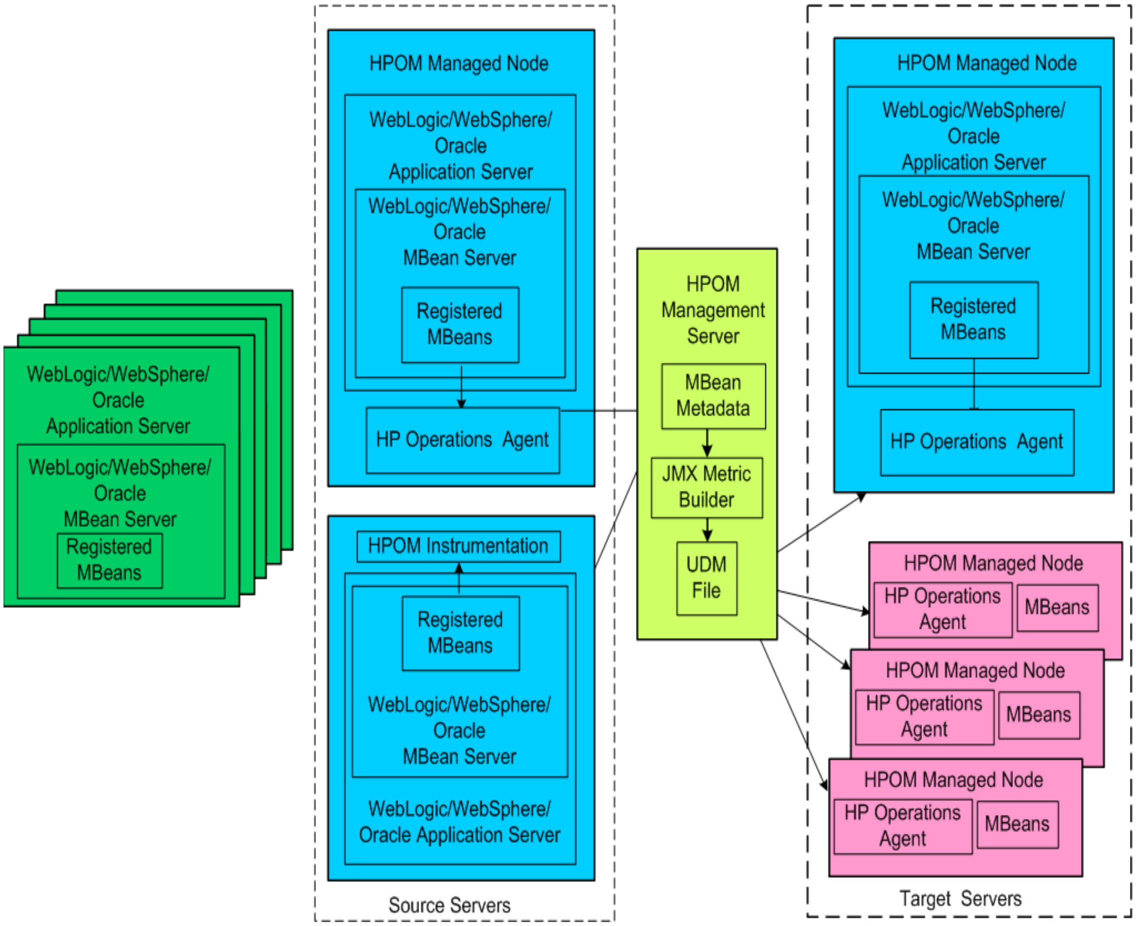
To monitor your applications using HP Operations Manager (HPOM), configure your MBean server environment and create policies to monitor and collect the data generated by the UDMs.

The MBean Server Environment

Using HPOM and the HP Operations Smart Plug-in for the supported Application servers, the HPOM management server can monitor servers whose MBeans are registered in the Application MBean Server.

The HPOM management server must be configured to gather MBean data (also referred to as metadata) from source servers and to monitor target servers using UDMs. The following figure illustrates the MBean Server Environment.

Figure 1 MBean Server Environment



Source servers are systems on which the Application MBean Servers are present. The HPOM instrumentation must be distributed to the source servers (the HPOM management server might not monitor them). The source servers can be an MBean staging area or development server.

The HPOM 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 HPOM management server. The target server can be a production server. Alarms, graphs, and reports are generated by UDMs based on MBeans registered in the supported Application servers.

Using the Application MBean Server

The Application servers include a built-in MBean server. Perform additional tasks such as installing the SPIJMB software, configuring the Application server SPIs to collect the MBean data, and using the JMX Metric Builder (an application that helps to create UDMs and browse MBeans) to create UDMs . These tasks are described in Chapter 2, Installing the SPI JMB and Chapter 3, Configuring the SPI JMB.

HPOM and Other Components

Additional components must be configured to create UDMs.

JMX Metric Builder

The JMX Metric Builder (JMB) is an application integrated with HPOM used to create UDMs that gather data from application MBeans registered in the supported Application servers. 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 HPOM policies. For more information about using the JMB, see the *JMX Metric Builder Online Help* or *JMX Metric Builder Online Help PDF* for JMX Metric Builder.

MBean data is obtained from a cache on the HPOM management server. You must run the Gather MBean Data tool to gather the MBean data that is stored in the cache on the HPOM management server. For more information about the Gather MBean Data tool and configuration requirements, see The Gather MBean Data Tool on page 13.

The JMB is installed with the SPIJMB software. For more information, see Install the SPIJMB Software on page 17.

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 HPOM environment. The JMB Plug-in for Eclipse is launched from Eclipse, not HPOM. The JMB Plug-in for Eclipse includes the same features as the JMB and can also test UDMs.

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

The JMB Plug-in for Eclipse is downloaded from the HPOM management server (you must install the SPIJMB software (for more information, see Install the SPIJMB Software on page 17).

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

The Gather MBean Data Tool

The Gather MBean Data tool gathers MBean information from selected managed nodes and enables 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, see Gather MBean Data Tool on page 26.

The Gather MBean Data tool is installed with the SPIJMB software. For more information, see Install the SPIJMB Software on page 17.

Metric and Collector Policies

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

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

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

Both policies must be defined and distributed to the target servers. For more information about these tasks, see Chapter 5, UDM Development.

2 Installing the SPI JMB

This chapter describes the procedure for installing the supported Application server SPI software, and the SPIJMB software.

You must install the SPI software and SPIJMB software, before you can develop UDMs using the JMX Metric Builder (JMB).

Figure 2 Flowchart on steps for installing and configuring the SPI

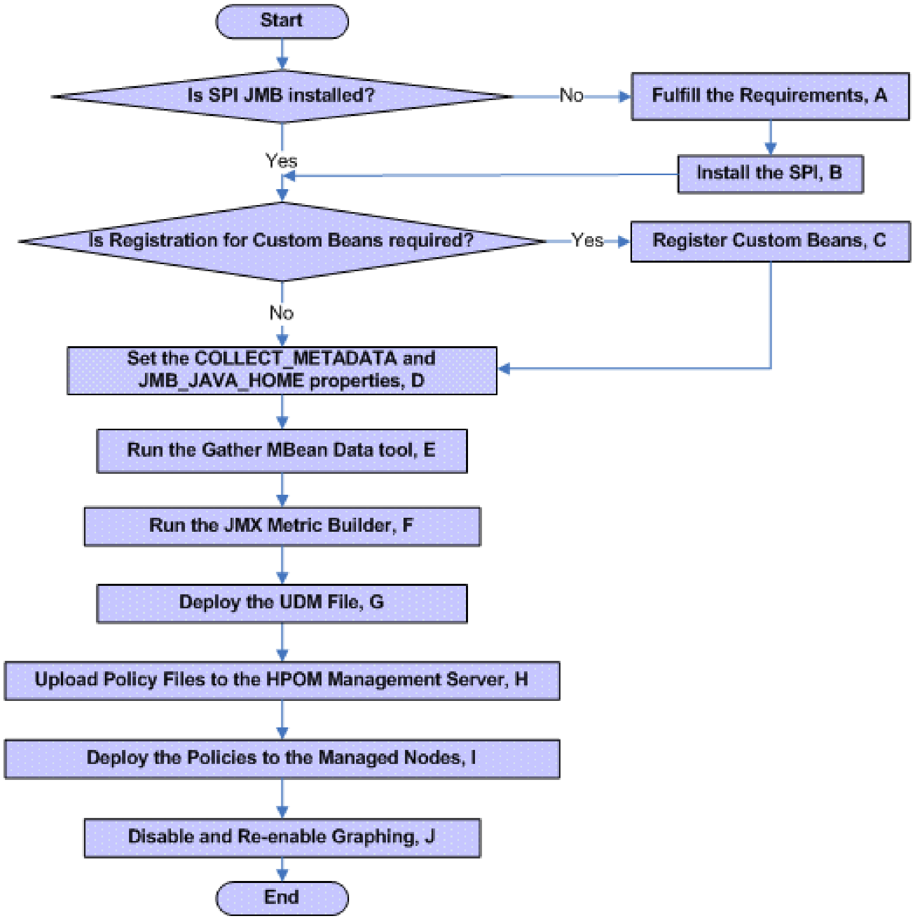

Start
Is SPI JMB installed?
No
Fulfill the Requirements, A
Yes
Install the SPI, B
Is Registration for Custom Beans required?
Yes
Register Custom Beans, C
No
Set the COLLECT_METADATA and JMB_JAVA_HOME properties, D
Run the Gather MBean Data tool, E
Run the JMX Metric Builder, F
Deploy the UDM File, G
Upload Policy Files to the HPOM Management Server, H
Deploy the Policies to the Managed Nodes, I
Disable and Re-enable Graphing, J
End

Table 1 References of thelegends in the flowchart

A	Built-in MBean Server Requirements on page 16
B	Install the SPIJMB Software on page 17
C	Register Custom MBeans on page 21
D	Set the COLLECT_METADATA and JMB_JAVA_HOME properties on page 22
E	Run the Gather MBean Data Tool on page 23

Table 1 References of thelegends in the flowchart

F	Run the JMX Metric Builder on page 32
G	Deploy the UDM File on page 34
H	Upload Policy Files to the HPOM Management Server on page 32
I	Deploy the Policies to the Managed Nodes on page 39
J	Disable and Re-enable Graphing on page 39

➤ For JMB Plug-in for Eclipse, you need to copy files, before deploying the UDM file (Legend Key - G). For more information, see Upload Policy Files to the HPOM Management Server on page 32.

Built-in MBean Server Requirements

If you are using the MBean server that is built into the supported Application server, you must install the following software:

- Supported Application Server SPI software
- SPI JMB

Installing the AS SPI Software

➤ Complete SPI software installation information is available in the respective SPI Installation and Configuration Guide.

For an HP-UX 11.31 IA management server, type:

```
swinstall -s /dvdrom/HPUX/HP_Operations_Smart_Plug-ins_HPUX.depot
<SPI>
```

where **`<SPI>`** is the supported Application Server SPI (WLSSPI/WBSSPI/OASSPI)

For a Solaris management server, type:

```
pkgadd -d /dvdrom/SOLARIS/HP_Operations_Smart_Plug-ins_SOLARIS.sparc
HPOvSpi<as>
```

where **`<as>`** is the supported Application server (wls/wbs/oas)

If you are using the MBean server that is built into the application server, you must configure the supported Application server SPI software. For more information, see the *Installation and Configuration guide for the <supported Application Server>*.

Install the SPIJMB Software

➤ The following examples show the command line usage of swinstall. For HP-UX systems, you can also use the graphical user interface (GUI).

Installing the SPI on the HPOM for HP-UX management server

For an HP-UX 11.31 IA management server, type:

```
swinstall -s /dvdrom/UNIX/HP_Operations_Smart_Plug-ins_HPUX.depot
SPIJMB
```

For a Solaris management server, type:

```
pkgadd -d /dvdrom/SOLARIS/HP_Operations_Smart_Plug-ins_SOLARIS.sparc
HPOvSpiJmb
```

Installing the SPI on the HPOM for Linux/Solaris management server

To install the SPI on the Linux/Solaris management server, perform any one of the following procedures:

- Installing the SPI through Graphical User Interface
- Installing the SPI through Command Line Interface

Installing the SPI through Graphical User Interface

To install the SPI using X-Windows client software, perform the following steps:

1. Login as a `root` user.
2. Insert the HP Operations Smart Plug-ins DVD into the DVD drive of the Linux management server. Mount the DVD if necessary.
3. Start the X-windows client software and export the DISPLAY variable by typing the following command:

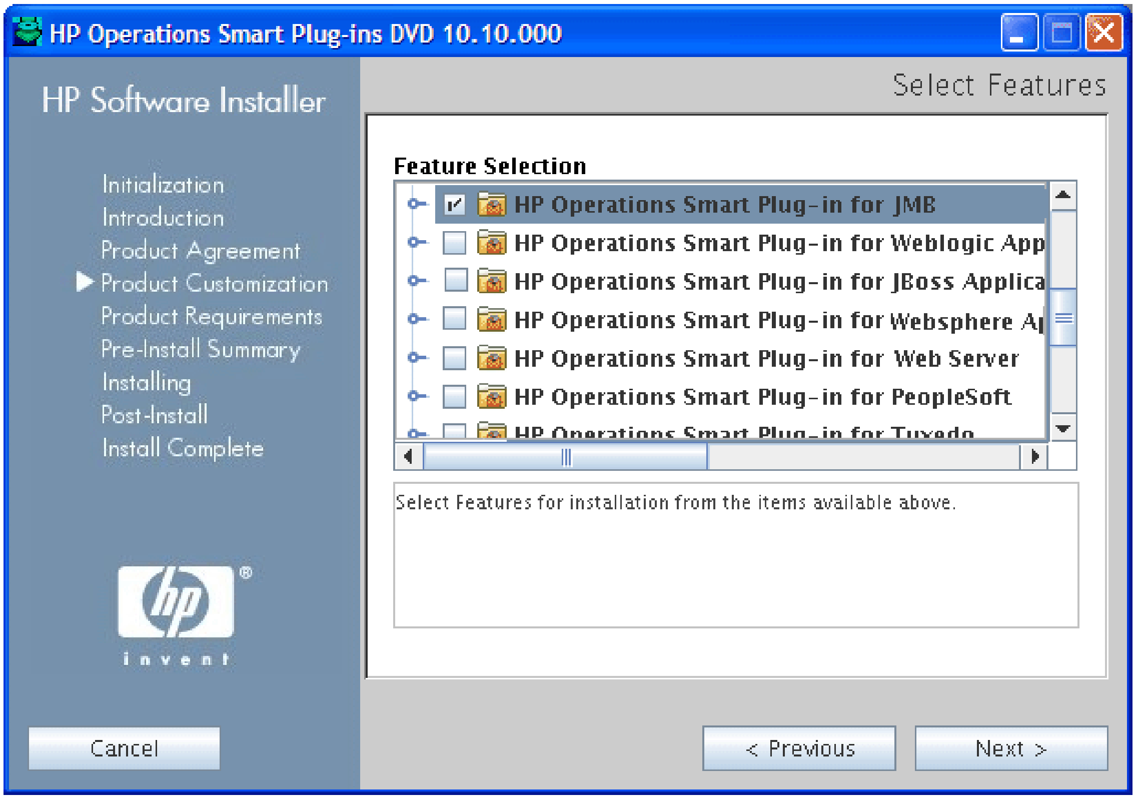
   ```
   export DISPLAY=<ip address>:0.0
   ```

4. To start the installation, type the following command:

   ```
   ./HP_Operations_Smart_Plug-ins_Linux_setup.bin
   ```

   The introductory window appears.
5. Select the language from the drop-down list and click **OK**. The Introduction (Install) window appears.
6. Click **Next**. The License Agreement window appears.
7. Select **I accept the terms of the License Agreement** button and click **Next**. The Select Features window appears.

8 Select the **HP Operations SPI for JMB** check box and click **Next**.



Feature Selection

- ☑ HP Operations Smart Plug-in for JMB
- □ HP Operations Smart Plug-in for Weblogic App
- □ HP Operations Smart Plug-in for JBoss Applica
- □ HP Operations Smart Plug-in for Websphere A
- □ HP Operations Smart Plug-in for Web Server
- □ HP Operations Smart Plug-in for PeopleSoft
- □ HP Operations Smart Plug-in for Tuxedo

Select Features for installation from the items available above.

By default, the HP Operations Smart Plug-in Common Components are selected.

The Install Check window opens.

9 Click **Next**. The Pre-Install Summary window opens.

10 Click **Install**.

While installing, you can see the Force reinstallation of already installed component packages check box. You can use either of the following options:

— Select the Force reinstallation of already installed component packages check box to reinstall the selected components, as applicable.

— Clear the Force reinstallation of already installed component packages check box to prevent reinstallation of the selected HP Software components, as applicable. Clearing the check box does not change the currently installed software components.

If the installation fails, you can quit installation. Click **Quit** to stop the installation. This does not uninstall the components installed till then.

The Installing window appears. The Install Complete window appears once the SPI is uninstalled.

11 Click **Done** to complete the installation.

Installing the SPI through Command Line Interface

To install the SPI through command line interface:

1 Login as a **`root`** user.

2 Insert the HP Operations Smart Plug-ins DVD into the DVD drive of the Linux management server. Mount the DVD if necessary.

3 To start the installation, type the following command:

```
./HP_Operations_Smart_Plug-ins_Linux_setup.bin -i console
```

4 When the prompt, ‘Choose Locale...’ appears, press the number corresponding to the language you want to choose.

5 Press **Enter** to continue. The Introduction screen appears.

6 Press **Enter** to continue.

7 When the prompt, ‘I accept the terms of the License Agreement’ for the License information appears, press **Y** to accept the terms and continue installation.

8 When the prompt, ‘Please select Features’ for the selection of the feature appears, press the number corresponding to the feature you want to install.

When you have installed one SPI on the Linux management server and want to install another SPI on the server, you have to reselect the previously installed SPI and select the required SPI from the Modify option. If you do not reselect the previously installed SPI, it removes the previously installed SPI and installs the selected SPI on the Linux management server.

9 Press **Enter**. A series of message appears. Follow the instructions as displayed in the message.

When the installation is complete, you will receive a message which states that the installation is completed successfully.

The SPIJMB software includes the following:

Item		Description	Location
Tools	Deploy UDM	Deploys the UDM file from the management server to the selected managed nodes.	JMX Metric Builder/ <SPI> tool group, where <SPI> refers to the corresponding supported Application server.
	Gather MBean Data	Gathers MBean information from the selected managed nodes.	
	JMX Metric Builder	Launches the JMB.	
	UDM Graph Enable/Disable	Starts/stops data collection for UDM graphs.	

3 Configuring the SPI JMB

This chapter describes the configurations needed by your environment and their procedure. 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 servers.

To configure your environment, complete the following:

- Register Custom MBeans (optional)
- Configure your MBean Server Environment (for more information, see Configure the MBean Server Environment on page 22)
- Complete Additional Configuration (for more information, see Additional Configuration on page 23)

Register Custom MBeans

Register your custom MBeans (optional), before configuring your MBean server environment. However, custom MBeans *must* be registered in the supported Application MBean server, if you want to monitor and collect data from them.

If you are using the WebLogic or Oracle MBean server, the Name attribute is used to identify its MBeans. If your MBean is a multi-instance MBean, 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, 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, each MBean instance must have a unique value in its mbeanIdentifier ObjectName key property. If the mbeanIdentifier ObjectName key property is not provided, 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.

See 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

Configure the MBean Server Environment

The node on which the supported Application MBean server is running must be configured so that MBean information can be gathered.

Supported Application MBean Server

If you are using the MBean server that comes with the Application Servers, your MBean server environment might look similar to the supported Application server.

To configure this MBean server environment:

1. Configure the supported Application server SPI

   Configure the Application server SPI software on the source and target servers. For more information, see *Chapter 3* of the *HP Operations Smart Plug-in for <Application Server Name> Installation and Configuration Guide*.

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

2. Set the COLLECT_METADATA and JMB_JAVA_HOME properties

   Set the COLLECT_METADATA server property of the source server to ON and the JMB_JAVA_HOME property to an installation of Java version 1.5 or later. This example shows the steps using the WebLogic SPI. If you have installed the supported Application server SPI, change any occurrence of WLSSPI to WBSSPI or OASSPI):

   a. From the HPOM console, select **Integrations** → **HPOM for Unix Operational UI**.

   b. Select one or more nodes on which you want to launch Discover or Configure WBSSPI tool.

   c. Right-click a node and select **Start** → **SPI for WebSphere** → **SPI Admin** → **Discover or Configure WBSSPI**.

      The Tool Selector window appears.

   d. Select the Launch Configure Tool radio button and click **OK**. The Introduction window appears.

   e. Click **Next**. The configuration editor appears.

   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 1.5 or higher for the management server. For more information about using the configuration editor, see Appendix B of the WebLogic SPI, WebSphere SPI or Oracle AS SPI Installation and Configuration Guide.

   g. Click **Next** to save the change and exit the editor. The Confirm Operation window appears.

   h. Click **OK**.

➤ If you click **Cancel** and make 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 the nodes, start the Discover or Configure WLSSPI tool, launch the Configure tool, click **Next** from the configuration editor, and then click **OK**.

3. Complete the Additional Configuration on page 23.

Additional Configuration

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

1. Run the Gather MBean Data Tool
2. Add a UDM Message Group
3. Assign the Message Group to an Operator

Run the Gather MBean Data Tool

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

➤ 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. See Supported Application MBean Server on page 22 for information on how to set the COLLECT_METADATA property.

To run the Gather MBean Data tool, follow these steps. This example shows the steps using the WBS SPI; if you installed the WLS/OAS SPI, change any occurrence of WLSSPI to WBSSPI or OASSPI:

1. From the Administration UI, select **Integrations** → **HPOM for Unix Operational UI**.
2. Select the node on which you want to run the Gather MBean Data tool.
3. Right-click on a node and select **Start** → **JMX Metric Builder** → **WLSSPI** → **Gather MBean Data.**

   The Gather MBean Data Output window appears.

For more information about this tool, see Gather MBean Data Tool on page 26.

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, see the *HP Operations for UNIX Concepts Guide*.

To add a message group:

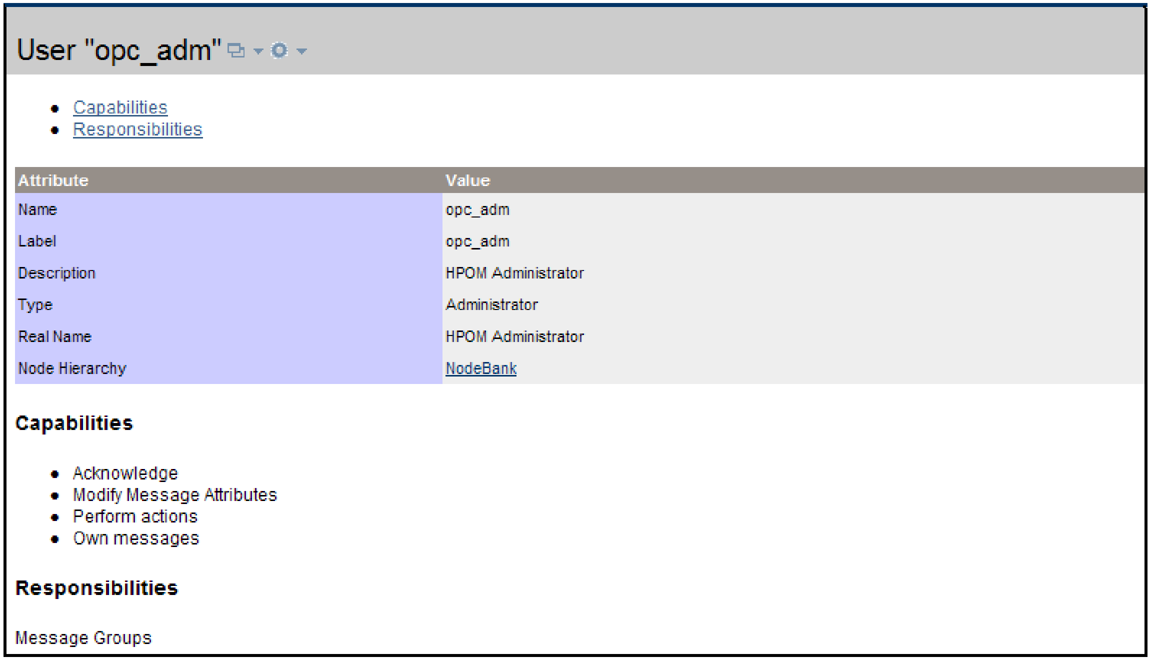
1. Open the All Message Groups window.
2. Select **Add Message group...** from the **Choose an Action...** drop-down list and click » to submit.
3. Enter a name (for example, Sales), label, and description.
4. Click **Save**. The message group is added.

Assign the Message Group to an Operator

To assign the message group to an operator:

1. Log on to HPOM as administrator.
2. Select **All Users** → **<Name of the operator>**. For example: **opc_adm**.

The User “opc_adm” screen appears.



User "opc_adm"

- Capabilities
- Responsibilities

Attribute	Value
Name	opc_adm
Label	opc_adm
Description	HPOM Administrator
Type	Administrator
Real Name	HPOM Administrator
Node Hierarchy	NodeBank

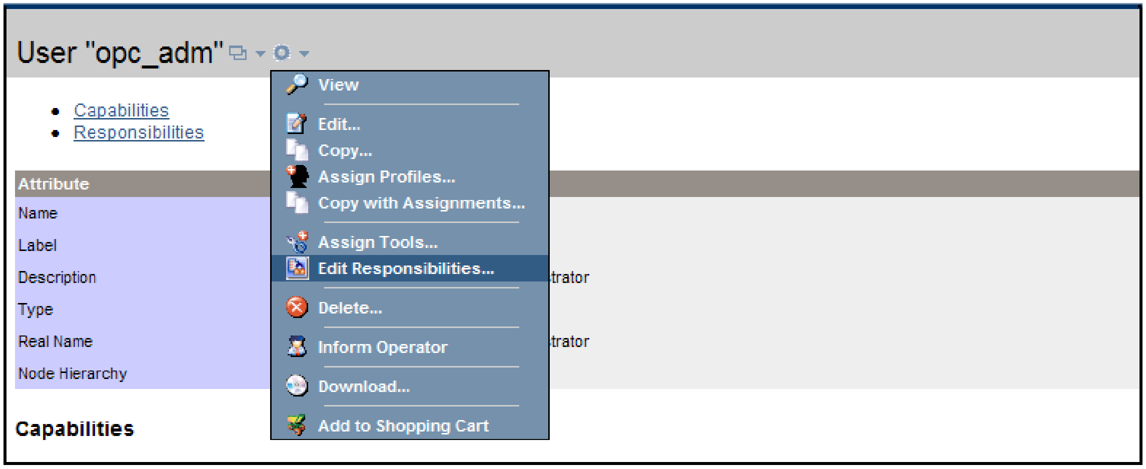
Capabilities

- Acknowledge
- Modify Message Attributes
- Perform actions
- Own messages

Responsibilities

Message Groups

3 To change a User’s responsibility, select **Edit Responsibilities...**. from the drop-down list as illustrated in the following figure.



User "opc_adm"

- Capabilities
- Responsibilities

Attribute	
Name	
Label	
Description	trator
Type	
Real Name	trator
Node Hierarchy	

Capabilities

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**. The message group is assigned to the operator.

4 Using the SPI JMB Tools

The SPI JMB offers centralized tools that help you to collect data from any of the supported Application servers.

Overview

The SPIJMB software contains the following tools:

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

Along with the tools installed with the SPIJMB software, the Admin tools of the Application server SPI can be run, even if you are not managing an Application Server.

- Discover or Configure the Application server SPI
- Self-Healing Info
- Start/Stop Monitoring
- Start/Stop Tracing
- Verify
- View Error File or View Error Log (for Oracle AS SPI (version 10gR3 only))

For more information on these tools, see *HP Operations Smart Plug-in for <Application Server name> Installation and Configuration Guide.*

➤ The examples in this chapter are for the WLS SPI. If you have installed the WBS/OAS SPI, replace any occurrence of WLSSPI with WBSSPI or OASSPI and WLS with WBS or OAS.

➤ WBSSPI/ WLSSPI/OASSPI Admin tools, not listed here, cannot be run successfully.

JMX Metric Builder Tool Group

The following tools are available in the WLS/WBS/OAS SPI tool group under the JMX Metric Builder tool group. These tools require the “root” user permission.

Deploy UDM Tool

Deploys the UDM file from the management server to the selected managed nodes. UDMs enable you to define your own metrics and monitor tools registered with the application MBean server.

Function

Deploy UDM deploys the UDM file from the management server to the following locations on the selected managed nodes.

For HP-UX, Solaris, AIX, Windows:

- `<%OVAgentDir%>/wasspi/wbs/conf/wasspi_<wbs/wls/oas>_udmDefinitions.xml`

For HP-UX, Solaris, AIX (non root HTTPS Agent environment):

- `%OVAgentDir%/conf/wbsspi/wasspi_<wbs/wls/oas>_udmDefinitions.xml`

All XML files in the `/opt/OV/conf/<wbsspi/wlsspi/oasspi>/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.
```

Gather MBean Data Tool

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

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

Setup

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

Function

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

Initially, the MBean information is saved in an XML file on the managed node at the following location.

For HP-UX, Solaris, AIX, Windows:

- `/var/opt/OV/wasspi/<wbs/wls/oas>/tmp/<NAME | ALIAS>.xml,`

For HP-UX, Solaris, AIX (non root HTTPS Agent environment):

- `/var/opt/OV/tmp/<wbs/wls/oas>/<NAME | ALIAS>.xml`

The NAME and ALIAS are the properties set for the managed node. The ALIAS property is always used if it is set.

After Gather MBean Data has collected the MBean information for a managed node, the MBean information is transferred to the HPOM management server and is saved in a cache file named:

- `/opt/OV/wasspi/<wbs/wls/oas>/metadata/`*<managed_node>*`/<NAME | ALIAS>.xml`

The XML file on the managed node is deleted. If a cache file on the HPOM management server is no longer needed, it is automatically deleted.

JMX Metric Builder Tool

Launches the JMB enabling you to edit the UDM file and browse MBeans on an MBean server. Run only one instance of the JMB at a time.

Setup

Complete the following tasks before running the JMB:

- Register your custom MBeans. For more information, see Register Custom MBeans on page 21.
- Configure you MBean server environment. For more information, see Configure the MBean Server Environment on page 22.
- Run the Gather MBean Data tool. For more information, see Run the Gather MBean Data Tool on page 23.

Function

JMX Metric Builder enables you to do the following:

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

UDMs for all HPOM managed nodes are maintained in UDM files on the HPOM management server. Use the Deploy UDM tool to distribute the UDM files from the management server to the managed nodes.

UDM Graph Enable/Disable Tool

This tool starts/stops data collection for UDM graphs and also starts/stops the HPOM subagent.

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

➤ Data logging does not work if the user defined metrics defined in the UDM `MetricsDefinition.xml` do not work. Hence, ensure that at least one of the user defined metrics (defined in the UDM `MetricsDefinition.xml`) work.

Function

UDM Graph Enable starts UDM data collection for graphing.

UDM Graph Disable stops UDM data collection for graphing.

Enable JMB Tracing

You can enable JMB tracing through the JMB GUI.

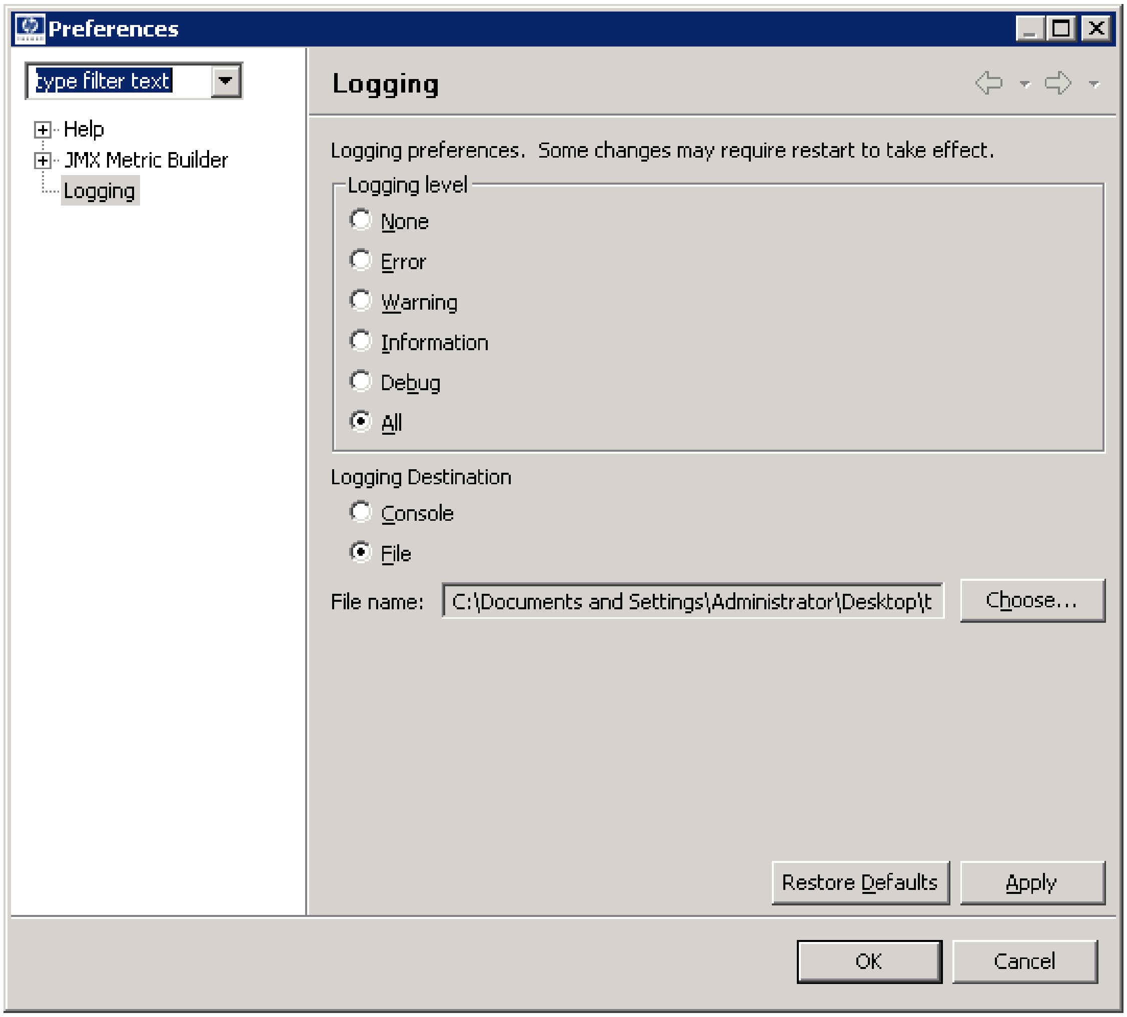
To enable JMB tracing:

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.



Logging

Logging preferences. Some changes may require restart to take effect.

Logging level

○ None
○ Error
○ Warning
○ Information
○ Debug
● All

Logging Destination

○ Console
● File

File name: C:\Documents and Settings\Administrator\Desktop\t

3 Click **Apply** and then click **OK**.

Launching Tools

To launch all tools:

1. From the Administration UI, select **Integrations** → **HPOM for Unix Operational UI**.
2. Select the nodes on which you want to launch the tool.
3. Right-click a node and select **Start** → **JMX Metric Builder** →<**WLSSPI**>→ ***<Name of the Tool>***.
   The <*Name of the Tool*> Output window appears.

5 UDM Development

This chapter explains how to create and monitor UDMs. After your source and target servers are configured and metadata is being collected from your MBean servers, you are ready to create and monitor UDMs.

To create and monitor UDMs:

1. Update Existing UDMs
2. Run the JMX Metric Builder
3. Add JMX Actions (Optional)
4. Upload Policy Files to the HPOM Management Server
5. Deploy the UDM File
6. Create a UDM Policy Group and Policies
7. Deploy the Policies to the Managed Nodes
8. Disable and Re-enable Graphing

Update Existing UDMs

If you have already created UDMs for the supported Application server:

1. If you want to use the JMB to edit existing UDMs (based on registered MBeans), update your alarming, graphing, and reporting UDMs to use a metric ID of <SPI>_1*xxx* or JMXUDM_1*xxx* (where <SPI> is the supported Application server and *xxx* is a number from 000 through 999).

   Do not update your existing UDMs with metric IDs <SPI>_07*xx (*where <SPI> is the supported Application server) . 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 or reporting metrics.

2. If you update your UDMs, update your policies to reflect the new metric IDs.
3. Move the existing UDM files on the management server so that they are deployed by the Deploy UDM tool to the managed nodes.

   If you are using the WebSphere 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 WebLogic 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>`.

If you are using the Oracle AS SPI (version 10gR3 only), move the file `/opt/OV/wasspi/oas/conf/wasspi_oas_udmDefinitions.xml` (and any other UDM file) to the directory `/opt/OV/wasspi/oas/conf/workspace/<UDMProject>`.

Run the JMX Metric Builder

This example shows the steps using the WLS SPI; if you have installed the WBS/OAS SPI, simply change any occurrence of WLSSPI to WBSSPI or OASSPI:

➤ If you did not convert your alarming, graphing, and reporting metrics to use metric IDs of <SPI>_1*xxx* or JMXUDM_1*xxx* (where <SPI> is the supported Application server and *xxx* is a number from 000 through 999) as described in step 1 of Update Existing UDMs on page 31, do not open this UDM file in the JMB.

To start the JMB:

1. From the Administration UI, select **Integrations** → **HPOM for Unix Operational UI**.
2. Select the nodes on which you want to run the JMX Metric Builder tool.
3. Right-click on the nodes and select **Start** → **JMX Metric Builder** → **WLSSPI** → **JMX Metric Builder.** Run only one instance of the JMB at a time.

For more information about JMB, see the *JMX Metric Builder online help* or *JMX Metric Builder Online Help PDF*.

Complete the following tasks to create a UDM using the JMB (for more information, see the *JMX Metric Builder online help* or *JMX Metric Builder Online Help PDF*):

1. Load metadata/MBean information
2. Organize MBeans (optional)
3. Open the UDM file
4. Add a metric
5. Change metric visibility (optional)

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

Add JMX Actions (Optional)

JMX actions are one or more JMX calls (invoke, get, set) performed on an MBean instance or type. Add JMX actions to a policy or metric. JMX actions cannot be added using the JMB. For information on adding JMX actions, see Appendix B, Add JMX Actions.

Upload Policy Files to the HPOM Management Server

If policies are generated using the JMB/JMB Plug-in for Eclipse, they must be uploaded to the HPOM management server.

To upload files on HP Operations Manager for HP-UX and Solaris:

a Copy the generated policy files from the development system to the following directories on the HPOM management server:

Files on Development System	Location on HPOM Management Server
<*User_Defined_Dir*>`/OMU/C/TEMPLATES/SCHEDULE/ext_metric_collect_schedule.dat`	<*Template_Dir*>`/C/TEMPLATES/SCHEDULE/`
<*User_Defined_Dir*>`/OMU/C/TEMPLATES/MONITOR/monitor.dat`	<*Template_Dir*>`/C/TEMPLATES/MONITOR/`
<*User_Defined_Dir*>`/OMU/C/TEMPLATES/TEMPLGROUP/policygroup.dat`	<*Template_Dir*>`/C/TEMPLATES/TEMPLGROUP/`

In this instance,

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

— <*Template_Dir*> is a user-specified directory on the HPOM management server. If you are only copying one set of policy files (a set of policy files consists of the `ext_metric_collect_schedule.dat`, `monitor.dat`, and `policygroup.dat` files), use the following:

`/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) or

`/var/opt/OV/share/tmp/OpC_appl/wasspi/udm/oas_set/` (Oracle) directory.

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

b Upload the policy information using the `opccfgupld` command. For example, type:
`/opt/OV/bin/OpC/opccfgupld -verbose -replace /opt/OV/wasspi/wls/conf/workspace/UDM/Output/OMU`

The policies are generated in the `/opt/OV/wasspi/wls/conf/workspace/UDM/Output/OMU` directory. If you copied more than one set of policy files to the HPOM management server, run this command for each set of policies. For more information on uploading configuration information, see the *HP Operations Developer's Toolkit Application Integration Guide*.

To upload files on HP Operations Manager for Windows:

1 Copy the generated policy files from "`<User_Defined_Dir>/OMW/`" on the development system to a temporary directory.

For example: `C:\temp\JMB\OMW`

2 From this directory, run the batch file : `uploadPolicies.bat`

For example: `C:/temp/JMB/OMW> uploadPolicies.bat`

Deploy the UDM File

Deploy the UDM file. Running the `Deploy UDM` tool creates a single UDM file from all XML files in the `/opt/OV/wasspi/<wbs/wls/oas>/conf/workspace/<UDMProject>/` directory.

This single UDM file is deployed on the managed nodes. For more information on this tool, see Deploy UDM Tool on page 26. This example shows the steps using the WebLogic SPI. If you installed the WebSphere SPI, or Oracle AS SPI (version 10gR3 only) change any occurrence of WLSSPI to WBSSPI or OASSPI.

1. From the Administration UI, select **Integrations** → **HPOM for Unix Operational UI**.
2. Select the node on which you want to run the Deploy UDM tool.
3. Right-click on a node and select **Start** → **JMX Metric Builder** → **WLSSPI** → **Deploy UDM.**

   The Deploy UDM Output window appears.

➤ If you are using the JMB Plug-in for Eclipse, you need to copy the files manually as described in Copy UDM File to the HPOM Management Server: For JMB, you only need to deploy the UDM file.

Copy UDM File to the HPOM Management Server:

Copy the UDM file from the development system to the following directories on the HPOM management server:

Files on Development System	**Location on HPOM Management Server**
<User_Defined_Dir>/*<UDM_File>*`.xml`	`/opt/OV/wasspi/<wbs/wls/oas>/conf/workspace/<UDMProject>/`
<UDM_Path>`/UDM/UDM`*<project>*`.xml` (JMB Plug-in for Eclipse)	`/opt/OV/wasspi/<wbs/wls/oas>/conf/workspace/<UDMProject>/`

In this instance,

- *<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/wls/oas>/conf/workspace/<UDMProject>/` on the HPOM management server must be uniquely named and end with `.xml`.

Create a UDM Policy Group and Policies

Creating a policy group for your UDMs enables you to assign multiple policies to a managed node as a single group rather than individually. Policies can be assigned to more than one policy group enabling you to customize the policies assigned to managed nodes.

Creating policies enables you to monitor your UDMs and define how often metrics are collected.

If you modify both the default and sample policy groups or either of them along with metric policies, your customizations are overwritten when you upgrade to the next version. However, if you copy or create a new policy group and policies, these customizations are *not* overwritten when you upgrade to the next version.

Create a Policy Group

If you are using the built-in MBean server that comes with the WebLogic/WebSphere/Oracle (version 10gR3 only) Application Server, the SPIs provide default policy groups and policies that you can copy. Copying an existing policy group or creating a new one enables you to keep custom policies separate from the original default policies.

To create a new policy group:

1. Open the Policy Bank Window.
2. Select **Add Policy Group** from the **Choose an Action...** drop-down list and click » to submit.
3. Fill in the text fields with appropriate information. For more information, see Naming the New Policy Group.
4. Click **Save** to save the changes. The new policy group is created.

Naming the New Policy Group

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

Create a Metric Policy

If you are using the built-in MBean server that comes with the supported Application Server, the SPIs provide default polices that you can copy.

To create a new policy:

1. Open the Policy Bank Window and select the policy group (parent policy group to which the newly created metrics policy will belong).
2. Select **Add Policy...** from the **Choose an Action...** drop down box and click » to submit.
3. Select the type of policy from the options present in the **Policy Type** drop down box.
4. Fill in the text fields with appropriate information.
5. Click **Save** to save the changes.

After you create a metric policy, you must name it, set conditions, and set threshold monitors.

Naming the Metric Policy

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

Setting Metric Conditions

To set metric conditions:

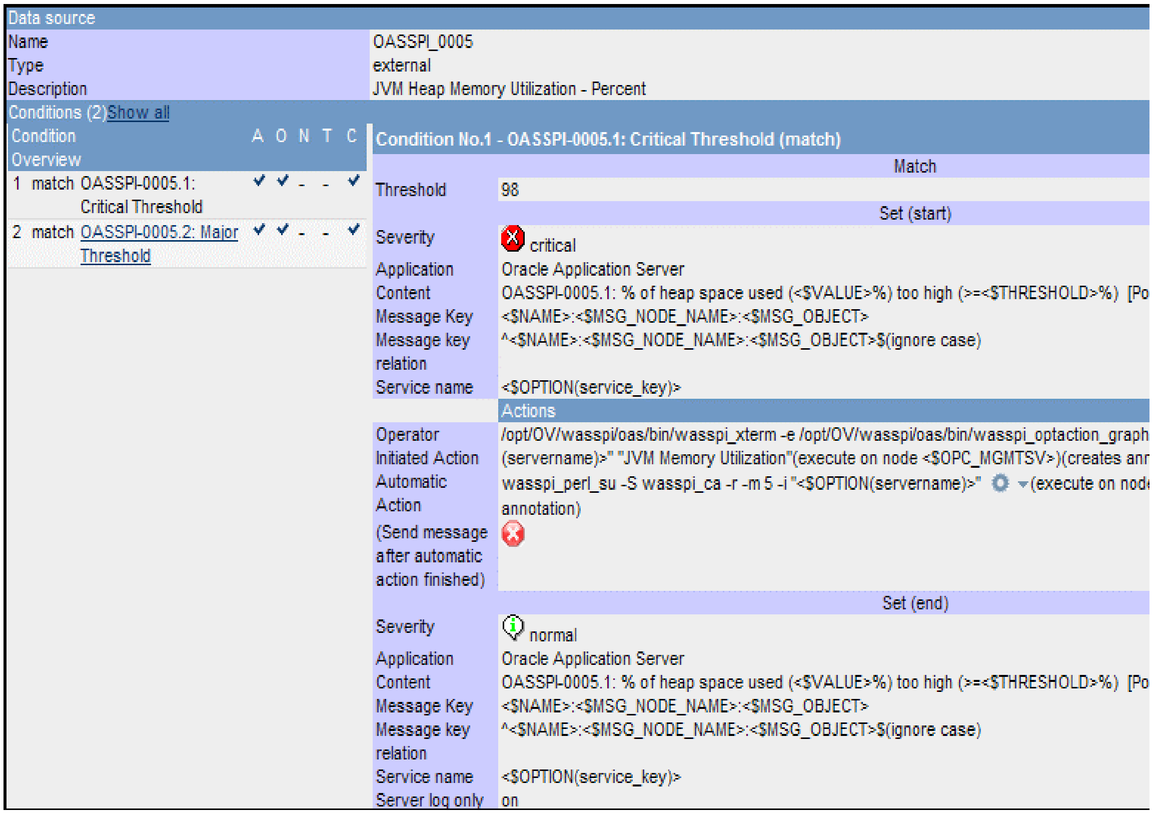
1. Open the Policy Bank window and click the parent policy group of the newly created metric.
2. Select the metric and click **Edit...** from the drop down box, which appears click ⚙ ▾.
3. The Edit Measurement_Threshold Policy “*Metric Name*” window appears.
4. Click the **Thresholds** tab and then click the condition you want to modify.

   The common items that you can 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 HPOM 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 Add a UDM Message Group on page 23.
   - **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 policy or condition. This response can be automatic or operator-initiated. This section provides the ability to generate Performance Manager graphs or reports, and 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 WebLogic 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 WebLogic SPI enables 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).
5. Click **Save**.
6. Distribute the policy as described in Deploy the Policies to the Managed Nodes on page 39.

The following figure shows a threshold setting of 95 for metric OASSPI-0005.1. This metric monitors the percentage of heap space used in the JVM. A value of more than 95 (but less than 98) for 20 minutes would generate an alarm (a message of the severity major)

Figure 3 Threshold Value for OASSPI 0005.1



Data source

Name	OASSPI_0005
Type	external
Description	JVM Heap Memory Utilization - Percent

Conditions (2) Show all

Condition Overview		A	O	N	T	C
1	match OASSPI-0005.1: Critical Threshold	✓	✓	-	-	✓
2	match OASSPI-0005.2: Major Threshold	✓	✓	-	-	✓

Condition No.1 - OASSPI-0005.1: Critical Threshold (match)

Match

Threshold	98

Set (start)

Severity	✖ critical
Application	Oracle Application Server
Content	OASSPI-0005.1: % of heap space used (<$VALUE>%) too high (>=<$THRESHOLD>%) [Po
Message Key	<$NAME>:<$MSG_NODE_NAME>:<$MSG_OBJECT>
Message key relation	^<$NAME>:<$MSG_NODE_NAME>:<$MSG_OBJECT>$(ignore case)
Service name	<$OPTION(service_key)>

Actions

Operator Initiated Action	/opt/OV/wasspi/oas/bin/wasspi_xterm -e /opt/OV/wasspi/oas/bin/wasspi_optaction_graph (servername)>" "JVM Memory Utilization"(execute on node <$OPC_MGMTSV>)(creates anr
Automatic Action	wasspi_perl_su -S wasspi_ca -r -m 5 -i "<$OPTION(servername)>" ⚙ ▾(execute on node annotation)
(Send message after automatic action finished)	✖

Set (end)

Severity	ⓘ normal
Application	Oracle Application Server
Content	OASSPI-0005.1: % of heap space used (<$VALUE>%) too high (>=<$THRESHOLD>%) [Po
Message Key	<$NAME>:<$MSG_NODE_NAME>:<$MSG_OBJECT>
Message key relation	^<$NAME>:<$MSG_NODE_NAME>:<$MSG_OBJECT>$(ignore case)
Service name	<$OPTION(service_key)>
Server log only	on

Setting Threshold Monitors

To set threshold monitors:

1 Open the Policy Bank window and open the UDM policy group.

2 Select a metric and click **Edit...** from the drop down box. The drop down box appears when you click ⚙ ▾.

3 The Edit Measurement_Threshold Policy window appears.

4 Click the **Thresholds** tab and then click the condition to modify the settings for message generation.

5 Modify the Message Generation settings by selecting the required option from the **Reset** drop-down list:

- **Use Same as threshold level**: 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.
- **Specify a special reset value...**: 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.

6 Click **Save** and then click **OK**.

7 Re-distribute the modified policies as described in Deploy the Policies to the Managed Nodes on page 39.

Create a Monitor Policy

If you are using the built-in MBean server that comes with the supported Application Server, the SPIs provide default policies which you can copy.

To create a new monitor policy:

1. Open the Policy Bank Window and select the policy group (parent policy group to which the newly created monitor policy will belong).
2. Select **Add Policy...** from the **Choose an Action...** drop-down list and click » to submit.
3. Select the type of policy from the options present in the **Policy Type** drop-down list.
4. Fill in the text fields with appropriate information.
5. Click **Save** to save the changes.

When you create a monitor policy, you must name it and set threshold monitors.

Naming and Setting Threshold Monitors for a Monitor Policy

To set name and threshold monitors:

1. Open the Policy Bank Window and select the parent policy group of the newly created monitor policy.
2. Select the collector policy to modify and click **Edit...** from the drop-down list, which appears when you click ⚙ ▾.
3. Enter the Name and the description accordingly.

   The name you give a copied collector policy 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 policy SALES-10m.

   The collector command of this collector policy must include the new name.
4. Click the **Scheduled Task** tab.
5. In the Command text box, enter the collector command (`wasspi_perl_su -S wasspi_ca -prod <wbs/wls/oas>` followed by these options:

Option	Description
`-c`	**Required**. The collector policy name (entered in the Monitor Name text box). Example: `-c SALES-10m`
`-m`	The metric numbers 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. For more information on this command, see the Using the Collector/Analyzer Command with Parameters section of the *HP Operations Smart Plug-in for <IBM WebSphere Application Server> Installation and Configuration Guide.*
6. Edit the polling interval.

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

7. Click the **Message Failed** tab.
8. Edit the message text. and then click **OK**.
9. Distribute the policy as described in Deploy the Policies to the Managed Nodes on page 39.

Syntax Examples

The examples that follow are for the WLS SPI. If you installed the WBS/OAS SPI replace any occurrence of wls with wbs or oas.

```
wasspi_perl_su -S wasspi_ca -prod <wbs/wls/oas> -c SALES-10min -m
1000-1005,1010
-x prefix=SalesUDM_
wasspi_perl_su -S wasspi_ca -prod <wbs/wls/oas> -c SALES-15min -m
1100-1120
-x prefix=SalesUDM_
```

Deploy the Policies to the Managed Nodes

To deploy policies to the managed nodes:

1. Open the All Node Groups window and select the node group.
2. Select **Deploy Configuration...** from the **Choose an Action** drop-down list and click » to submit.
3. Select **Distribute Policies** and then click **OK**. The policies are now distributed to the selected node group.

Monitors can now begin running according to their specific collection interval.

Disable and Re-enable Graphing

The supported Application server SPIs can be used with HP Performance Manager to generate graphs showing the collected metric values. To collect the metric values of the UDMs you have created, restart the data collection by disabling and enabling graphing.

1. If graphing is enabled, disable it:
   - a. From the Administration UI, select **Integrations** → **HPOM for Unix Operational UI**.
   - b. Select the node on which you want to disable graphing.
   - c. Right-click on the node and select **Start** → **JMX Metric Builder** → **WLSSPI** → **UDM Graph Disable.** The UDM Graph Disable Output window appears.
2. From the SPI, enable graphing
   - a. From the Administration UI, select **Integrations** → **HPOM for Unix Operational UI.**
   - b. Select the node on which you want to enable graphing.
   - c. Right-click on the node and select **Start** → **JMX Metric Builder** → **WLSSPI** → **UDM Graph Enable.** The UDM Graph Enable Output window appears.

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

6 Removing the SPI JMB

This chapter provides details on how to remove the SPI JMB components.

Removing the SPI components

If you do not want to create or use UDMs, and do not want to use the JMX Metric Builder, complete the following tasks:

1. Remove Software from the Management Server
2. Delete Custom Message Groups

Remove Software from the Management Server

1. Open a terminal window and log on as root.
2. In the terminal window, enter the following:
   - For an HP-UX 11.31 IA management server, type:

     ```
     /usr/sbin/swremove SPIJMB
     ```

   - For a Solaris management server, type:

     ```
     /usr/sbin/pkgrm HPOvSpiJmb
     ```

The **swremove** and **pkgrm** command removes the files from the software system, categories, node groups, tools, and policies.

To remove the SPI JMB through the Graphical User Interface from the Linux Management Server, using X-Windows client software, perform the following steps:

1. Login as a **root** user.
2. Insert the HP Operations Smart Plug-ins DVD into the DVD drive of the Linux management server. Mount the DVD if necessary.
3. Start the X-windows client software and export the DISPLAY variable by typing the following command:

   ```
   export DISPLAY=<ip address>:0.0
   ```

4. To start the removal of the SPI, type the following command:

   ```
   ./HP_Operations_Smart_Plug-ins_Linux_setup.bin
   ```

   The introductory window appears.
5. Select the language from the drop-down list and click **OK**. The Application Maintenance window appears.

6 Select **Uninstall** button and click **Next**. The Pre-Uninstall Summary window appears

➤ When you have two SPIs installed on the Linux management server and you want to remove one SPI out of the two installed SPIs, select Modify option and then the SPI you want to retain. Do not select the SPI which you want to remove.

7 Click **Uninstall**. The Uninstalling window appears. The Uninstall Complete window appears once the SPI is uninstalled.

8 Click **Done** to complete the removal of the SPI.

To remove the JMB through the Command Line Interface:

1 Login as a `root` user.

2 Insert the HP Operations Smart Plug-ins DVD into the DVD drive of the Linux management server. Mount the DVD if necessary.

3 To start the removal of the SPI, type the following command:

```
./HP_Operations_Smart_Plug-ins_Linux_setup.bin -i console
```

4 When the prompt, 'Choose Locale...' appears, press the number corresponding to the language you want to choose.

5 Press **Enter** to continue. The Maintenance Selection screen appears.

6 Press the appropriate option (number) to start the removal of the SPI.

➤ When you have two SPIs installed on the Linux management server and you want to remove one SPI out of the two installed SPIs, select Modify (1) option and then the SPI you want to retain. Do not select the SPI which you want to remove.

7 Press **Enter** to continue. When the removal is complete, you will receive a message which states that the removal is completed successfully.

Delete Custom Message Groups

1 Open the All Message Groups window.

2 Select the message groups for the custom groups by selecting the check box.

3 Select **Delete...** from the **Choose an Action** drop-down list and click » to submit.

The message groups for the custom groups are deleted.

7 Troubleshooting

This chapter provides information on basic troubleshooting.

JMB Usage

- **Problem:** JMB cannot read a UDM file that is broken, even if the problem with the file has been corrected. If the UDM file becomes corrupted and the JMB attempts to read that file, an error message is displayed. The file may become corrupted by manual editing. If the file is fixed, the JMB does not refresh and read the corrected file.

  Solution: The JMB should be exited. The errant file should be corrected or removed. Then, the JMB should be restarted. When it starts, the JMB can read the file or a new file with the same name can be created. The UDM files are stored in the following directory:

  ```
  /opt/OV/wasspi/<wbs/wls/oas>/conf/workspace/<UDMProject>/
  ```

- **Problem:** JMB cannot read an MBean data file that is corrupted. On rare occasions an error has occurred in the Gather MBean process and the MBean data file has been corrupted, either by being truncated or by containing an error message instead of the MBean XML. The error cannot be cleared, even by removing the bad file.

  Solution: There are two alternatives depending on what result you want. In either case, first exit the JMB. The simplest is to re-run the Gather MBean Data application to get the data successfully. The corrected data file will be properly read when the JMB is restarted.

  The second option is used if you do not want to create MBeans for the application server whose SPI MBean data file is bad. After exiting the JMB, remove the following directory and all of its contents: `/opt/OV/wasspi/udm/lib/ovojmxtool/configuration/`. Also, remove the bad MBean data file named:

  ```
  /opt/OV/wasspi/<wbs/wls/oas>/conf/workspace/<UDMProject>/<node-name>/
  <server-name>.xml
  ```

  This causes the JMB to forget the associations it had with previous files. When it starts, it only notes the files present in the MBean data directory.

- **Problem:** Application server dat files related to UDMs do not get updated on the managed node and show as 0 KB .dat file.

  Solution: The UDM dat files do not get populated with values, as the MBeans cannot be located for any metrics defined in the `udmMetricDefinition` file. The dat files will not be updated, if data is not available for at least one metric from the MBean for the WASSPI Collector. The MBean name specified in the `udmMetricsDefinitions.xml` should exist on the MBean Server. Ensure that at least one of the metrics work in order to log the value into the dat file.

- **Problem:** If the Gather Metadata's output xml file is huge (around 10 MB size), the JMB's MBean Explorer does not list the Mbeans and the following exception is thrown: (`"SAXParserException : Parse has reached the entity expansion limit of "64,000" set by the application"`).

Solution: Modify the following parameters in the launcher_udmbuilder script. This script is present in the jmb/bin/<spi> folder.

a Increase the heap size of JVM : (eg : “-Xms32m –Xmx256m”)

b Set “-DentityExpansionLimit=1000000”, to increase the xml parser’s expansion limit.

For example:

```
@cmd = (
"\"$Java\"",
"-Xms32m",
"-Xmx256m",
"-DentityExpansionLimit=1000000",
"-cp", "\"$classpath\"",
"org.eclipse.core.launcher.Main",
"-product", "com.hp.ovojmxtool.product",
"-SPI", "wlsspi",
"-consolelog",
"-data", "\"$workspace\"",
"-metadata", "\"$metadata\"",
) ;
```

Launch JMB

- **Problem:** You cannot start more than one instance of JMB at the same time.

  Solution: Run only one instance of JMB at a time.

Gather Metadata

- **Problem:** The Gather MBean Data application hangs.

  Solution: The MBean information is saved in an XML file on the managed node. Gather MBean Data application retrieves information from the file immediately, if the XML file is 1 KB in size. If the file size is larger (for example, 3 MB) then Gather MBean Data application may take about 20 minutes to retrieve the information.

- **Problem:** Gather Metadata fails due to timeout on the node in a WebSphere network deployer scenario.

  Solution: On the node, perform the following steps:

  a In `<OV_DATADIR>/wasspi/wbs/conf/SPIConfig` file add the following property :

  com.hp.openview.wasspi.collector.rmi.client.socket.timeout=<value in seconds>

  (preferably 10800 seconds, that is, 3 hrs).

  b In the instrumentation dir, `/var/opt/OV/bin/instrumentation`, run :

  wasspi_perl wasspi_ca -prod wbs -r -x metadata=on -i "<Server Name>".

  <Server Name>.xml in `/var/opt/OV/tmp/wbsspi` is created.

c. On the server, perform the following task:

   Copy the output of step c to `/opt/OV/wasspi/wbs/conf/metadata/<fully qualified node name>/<Server Name>.xml`.

   For example: `/opt/OV/wasspi/wbs/conf/metadata/btovm111.hp.com/<Server Name>.xml`

Deploy UDM

- **Problem:** Deploy UDM fails with xml validation errors.

  Solution: The deploy UDM tool fails while trying to generate the `udmMetricDefinition.xml` file if the UDM Workspace directory (`/opt/OV/wasspi/<wbs/wls/oas>/conf/workspace/UDM`) has several xml files (which were created out of JMB) with same metric IDs. Ensure that all the xmls in the “workspace/UDM” directory, do not have any duplicate IDs (that is metric numbers).

A Metric Definitions DTD

This appendix contains Metric Definition DTD and sample XML Files. It also describes 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 Application SPIs 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 following sections require familiarity with XML and DTD.

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

Operating System	Directory
UNIX or AIX	`/var/opt/OV/wasspi/<wbs/wls/oas>/conf/`
UNIX or AIX (new non-root HTTPS managed node)	`/var/opt/OV/conf/<wbsspi/wlsspi/oasspi>/`
Windows	`%OvAgentDir%\wasspi\<wbs/wls/oas>\conf\`

➤ You should not edit, rename or move the DTD files because they are used at runtime.

The following is a list of elements described in this appendix and the element hierarchy. An element's attributes 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
        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 this 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 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

This section provides a sample metric definition document to illustrate how you can 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. 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/jmb/samples/wasspi_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.

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 49, for version 6.1 and later, 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, see 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 can 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 this example, a string value collected by the collector (“Running”, “Shutdown Pending”) is mapped to an integer (1, 2). This integer value is used by HPOM policies to generate alarms/ messages. See Sample Metric Definition Document on page 51.

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 this example, the collector calculates the value of a metric (see Sample 1 on page 49) 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.
- Usable operands are metric IDs and literal doubles.

A metric ID can see an MBean metric and another calculated 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 49.

```
<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.

- If a FromVersion element is not present, no lower limit exists to the server versions supported by this metric.
- 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.

- If a ToVersion element is not present, no upper limit exists to the server versions supported by this metric.
- 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, and MBean 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 49, 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 50.

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 this example, the MBean operation `stagingEnabled` is invoked by passing the string parameter `examplesServer`. See Sample 2 on page 50.

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>
```

This example indicates that one JMX call will be performed on an MBean. See Sample 2 on page 50.

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 50 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 this example, for MBean `*:*, Type=JMSServerConfig`, the operation `stagingEnabled` is invoked by passing the string parameter `examplesServer`. See Sample 2 on page 50

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>
```

This example indicates that the string value “Running” has been mapped to the integer “1”. This integer value will be used by the HPOM policies to generate messages or 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>
```

This 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 later. Also, see Sample Metric Definition Document on page 51.

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+
      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. 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 and calculations. 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+
    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+),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 57) 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>
```

This example indicates that the value obtained from the formula, `JMSServerConfig_MessagesMaximum + (5-5)` will be an integer. See Sample 2 on page 50.

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>
```

This example indicates that `*:*,Type=ExecuteQueue` is the JMX-compliant object name of the MBean. See Sample 1 on page 49.

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>
```

This 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 50.

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 Parameter elements do not contain any attributes

Syntax

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

Example

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

This example indicates that a string parameter “examplesServer” is passed for an operation. See Sample 2 on page 50.

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>
```

This 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 50.

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"/>
```

This example indicates that a string value `examplesServer` is used by an operation. Sample 2 on page 50

ToVersion Element

See FromVersion and ToVersion Elements on page 58 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 50.

B 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:

- Using the Collector Command Parameters – Describes the collector command parameters and how to define a single JMX call using the collector command
- Defining JMX Actions in XML – Explains how to define and implement JMX actions in an XML file
- Defining JMX Actions in a Metric Definition – Explains 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) Enables you to list specific servers on which to perform the JMX actions. If this parameter is not specified, the JMX actions are performed on all configured servers. **Syntax**: `-i` *<server_name>* **Example**: `-i server1,server3`
`-m`	(metric) Specifies the metric ID containing the JMX actions to perform. This metric ID must be defined in a UDM file. This option must not be used with the `-mbean` or `-xml` options. **Syntax**: `-m` *<metric_id>* **Example**: `-m TestUDM_1000`

Parameter	Description	
`-mbean`	Performs a JMX call on the specified MBeans. This option must not be used with the `-m` or `-xml` options. **Syntax**: `-mbean` <*objectname*> <*action*> **Example**: `-mbean WebSphere:type=ThreadPool,* -set growable true` In this instance, <*action*> (a JMX call) is one of the following:	
	`-get`	Returns the value of the specified attribute. Syntax: `-mbean <objectname> -get <attribute>` **Example**: `-get maximumSize`
	`-invoke [-type]`	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*>`]...` In this instance, <*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`
	`-set`	Assigns the specified value to the specified attribute. **Syntax**: `-mbean` <*objectname*> `-set` <*attribute*> <*value*> **Example**: `-set growable true`
`-o`	(object) Specifies an MBean instance. **Syntax**: `-o` <*mbean_instance*> **Example**: `-o exampleJMSServer`	
`-xml`	Specifies the XML file that contains the JMX actions to perform (include the fully-qualified path). This option must not be used with the `-m` or `-mbean` options. **Syntax**: `-xml` <*filename*> **Example**: `-xml /tmp/myJMXActions.xml`	

WebSphere 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 (in this instance, `<$OPTION(instancename)>` specifies an alarming instance):

  ```
  wasspi_perl_su -S wasspi_ca -prod wbs -a -mbean
  WebSphere:type=ThreadPool,* -set maximumSize 500 -o
  <$OPTION(instancename)>
  ```

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

  ```
  wasspi_perl_su -S wasspi_ca -prod wbs -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_perl_su -S wasspi_ca -prod wbs -a -mbean
  WebSphere:type=ThreadPool,* -set maximumSize 50 -i server1
  ```

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

  ```
  wasspi_perl_su -S wasspi_ca -prod wbs -a -mbean
  WebSphere:type=ThreadPool,* -set maximumSize 50 -i server1 -o
  MessageListenerThreadPool
  ```

- Invoke an operation on a specific MBean instance:

  ```
  wasspi_perl_su -S wasspi_ca -prod wbs -a -mbean WebSphere:type=Perf,*
  -invoke setInstrumentationLevel pmi=m true -i server1 -o PerfMBean
  ```

- Get the ThreadPool maximumSize attribute:

  ```
  wasspi_perl_su -S wasspi_ca -prod wbs -a -mbean
  WebSphere:type=ThreadPool,* -get maximumSize -i server1
  ```

WebLogic 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 (in this instance, `<$OPTION(instancename)>` specifies an alarming instance):

  ```
  wasspi_perl_su -S wasspi_ca -prod wls -a
  -mbean "PetStore:*,Type=ExecuteQueueConfig"
  -set ThreadsMaximum 50 -o <$OPTION(instancename)>
  ```

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

  ```
  wasspi_perl_su -S wasspi_ca -prod wls -a -mbean *:*,Type=JMSServerConfig
  -set MessagesMaximum 250000 -i examplesServer
  ```

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

  ```
  wasspi_perl_su -S wasspi_ca -prod wls -a -mbean *:*,Type=JMSServerConfig
  -set MessagesMaximum 250000 -i examplesServer -o examplesJMSServer
  ```

- Invoke an operation on multiple MBean instances:

  ```
  wasspi_perl_su -S wasspi_ca -prod wls -a -mbean *:*,Type=ApplicationConfig
  -invoke staged -i examplesServer
  ```

- Get the MessagesMaximum attribute:

```
wasspi_perl_su -S wasspi_ca -prod wls -a -mbean *:*,Type=JMSServerConfig
-get MessagesMaximum -i examplesServer
```

Oracle AS SPI (version 10gR3 only) Command Line Examples

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

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

  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a -mbean *:*,j2eeType=ThreadPool
  -set maxPoolSize 40 -i home
  ```

- Get the ThreadPool maximumSize attribute:

  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a -mbean *:*,j2eeType=ThreadPool
  -get maxPoolSize -i home
  ```

- Set the maximum time in seconds that the data source will wait while attempting to connect to a database:

  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a -mbean
  *:*,j2eeType=JDBCDataSource
  -set loginTimeout 200 -i home
  ```

- Get the maximum time in seconds that the data source will wait while attempting to connect to a database:

  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a -mbean
  *:*,j2eeType=JDBCDataSource
  -get loginTimeout -i home
  ```

- Invoke an operation to set the value of a given system property:

  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a -mbean *:*,j2eeType=JVM
  -invoke setproperty key="TestVariable" value="test1" -i home
  ```

- Invoke an operation to return the value of a given system property:

  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a -mbean *:*,j2eeType=JVM
  -invoke getproperty key="TestVariable" -i home
  ```

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 82 for example XML files).
2. Copy and rename a monitor policy.
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 policy.

XML File Examples

- The following is an example XML file for WebSphere SPI (available online in `/var/opt/OV/wasspi/wbs/conf/JMXActions-sample.xml` or `/var/opt/OV/conf/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
wasspi_wbs_UDMMetrics-sample
           Therefore, UDM configuration needs to specify
wasspi_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=1"/>
        </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 WebLogic SPI (available online in `/var/opt/OV/wasspi/wls/conf/JMXActions-sample.xml` or `/var/opt/OV/conf/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
         wasspi_wls_UDMMetrics-sample Therefore, UDM configuration needs
to specify
         wasspi_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>
```

- The following is an example XML file for Oracle AS SPI (version 10gR3 only) available online in `/var/opt/OV/wasspi/oas/conf/JMXActions-sample.xml` or `/var/opt/OV/conf/oas/JMXActions-sample.xml`:

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

<!-- @WHAT_STRING@ -->

<!-- Sample JMX Actions XML -->

<JMXActions>

 <JMXAction>
  <JMXCalls>
    <ObjectName>*:*,j2eeType=ThreadPool</ObjectName>
    <!-- Set a new value.
         ThreadPool_maxPoolSize is defined in UDM file
         wasspi_oas_UDMMetrics-sample
         Therefore, UDM configuration needs to specify
wasspi_oas_UDMMetrics-sample.
    -->
    <Set>
      <Attribute>maxPoolSize</Attribute>
      <Value>
        <Numeric>
          <Formula>ThreadPool_poolSize + (5)</Formula>
        </Numeric>
      </Value>
    </Set>
    <Get>
      <Attribute>maxPoolSize</Attribute>
    </Get>
  </JMXCalls>
 </JMXAction>
  <!-- The following action demonstrates an operation invoke.
  -->
 <JMXAction>
  <JMXCalls>
    <ObjectName>*:*,j2eeType=JVM</ObjectName>
    <!-- Invoke an operation to set the value of a given system
         property  : For demonstration only.
    -->
    <Invoke>
      <Operation>setproperty</Operation>
      <Parameters>
        <Parameter>
          <String key="TestVariable"/>
        </Parameter>
        <Parameter>
          <String value="test1"/>
        </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_perl_su -S wasspi_ca -prod wbs -a
  -xml /var/opt/OV/wasspi/wbs/conf/JMXActions-sample.xml
  -i examplesServer
  ```

-
  ```
  wasspi_perl_su -S wasspi_ca -prod wls -a
  -xml /var/opt/OV/wasspi/wls/conf/JMXActions-sample.xml
  -i examplesServer
  ```

-
  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a
  -xml /var/opt/OV/wasspi/oas/conf/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 86 for example UDM files).
2. Copy and rename a collector policy.
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 policy.

UDM File Examples

- The following are example metrics for WebSphere SPI (available online in the /opt/OV/jmb/samples/wasspi_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">
  <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 which 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">
  <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 WebLogic SPI (available online in the** `/opt/OV/jmb/samples/wasspi_wls_UDMMetrics-sample.xml` **file):**

```
<!-- The Following metric defines a JMX action which 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>
```

- The following are example metrics for Oracle AS SPI (version 10gR3 only) (available online in the `/opt/OV/jmb/samples/wasspi_oas_UDMMetrics-sample.xml` file):

```
<?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="OASSPI_0100" name="ThreadPoolWaitCnt" alarm="yes">
      <MBean instanceType="multi">
        <FromVersion server="10.1" update="3" />
        <ObjectName>*:*,j2eeType=ThreadPool</ObjectName>
        <Attribute>queueSize</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.
    -->
    <Metric id="JVM_HeapFreeCurrent" alarm="no">
      <MBean instanceType="single">
        <FromVersion server="10.1" update="3" />
        <ObjectName>*:*,Type=JVM</ObjectName>
        <Attribute>freeMemory</Attribute>
      </MBean>
    </Metric>
    <Metric id="JVM_HeapSizeCurrent" alarm="no">
      <MBean instanceType="single">
        <FromVersion server="10.1" update="3" />
        <ObjectName>*:*,Type=JVM</ObjectName>
        <Attribute>totalMemory</Attribute>
      </MBean>
    </Metric>
    <!-- The following metric illustrates a calculated metric. The
    calculation is based on the previous 2 "base" metrics.
    -->
    <Metric id="OASSPI_0101" name="JVMMemUtilPct" alarm="yes" graph="yes">
      <Calculation>
        <FromVersion server="10.1" update="3" />
        <Formula>((JVM_HeapSizeCurrent-JVM_HeapFreeCurrent)/
JVM_HeapSizeCurrent)*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 policy.
    that the 'datatype' must be specified as 'string'.
    -->
    <Metric id="OASSPI_0102" name="State" alarm="yes" report="no">
      <MBean dataType="string">
        <ObjectName>*:*,Type=J2EEServer</ObjectName>
        <Attribute>eventProvider</Attribute>
        <AttributeValueMapping>
          <Map from="true" to="1" />
        <Map from="false" to="2" />
          </AttributeValueMapping>
      </MBean>
    </Metric>
```

```
    <!-- Metric IDs that are referenced from the collector command line
    must have a prefix followed by four digits. The default prefix is
    'WLSSPI_'.  The 'prefix' option must be used on the command line for
    the following metric since this metric has a different prefix than
    'WLSSPI_'. Example: wasspi_wls_ca -c FIRST_CLIENT_60-5MIN -x
    prefix=Testing_ -m 792 ...
    -->
    </Metric>
    <Metric id="Testing_0103" alarm="no">
      <MBean>
        <ObjectName>*:*,Type=J2EEServer</ObjectName>
        <Attribute>node</Attribute>
      </MBean>
    </Metric>
    <!-- This metric is used in a subsequent JMX action calculation.
    -->
    <Metric id="ThreadPool_poolSize">
      <MBean instanceType="multi">
        <ObjectName>*:*,Type=ThreadPool</ObjectName>
        <Attribute>poolSize</Attribute>
      </MBean>
    </Metric>
    <!-- The Following metric defines a JMX action which 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=ThreadPool</ObjectName>
        <Attribute>queueCapacity</Attribute>
      </MBean>

      <JMXActions>
        <JMXAction>
          <JMXCalls>
            <ObjectName>*:*,j2eeType=ThreadPool</ObjectName>
            <!-- Set a new value.
            ThreadPool_poolSize is defined in UDM file
wasspi_oas_UDMMetrics-
            sample
            Therefore, UDM configuration needs to specify
wasspi_oas_UDMMetrics-
            sample
            -->
            <Set>
              <Attribute>maxPoolSize</Attribute>
                <Value>
                <Numeric>
                  <Formula>ThreadPool_poolSize + (5)
                  </Formula>
                </Numeric>
                </Value>
             </Set>
             <Get>
               <Attribute>maxPoolSize</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=J2EEApplication</ObjectName>
        <Attribute>applicationRootDirectoryPath</Attribute>
      </MBean>
      <JMXActions>
        <JMXAction>
          <JMXCalls>
            <ObjectName>*:*,j2eeType=JVM</ObjectName>
            <!-- Invoke an operation to set the value of a given system
            property  : For demonstration only.
            -->
            <Invoke>
              <Operation>setproperty</Operation>
                      <Parameters>
                        <Parameter>
                          <String key="TestVariable" />
                        </Parameter>
                        <Parameter>
                          <String value="test1" />
                        </Parameter>
                      </Parameters>
            </Invoke>
          </JMXCalls>
        </JMXAction>
      </JMXActions>
    </Metric>
  <Metrics>

<MetricDefinitions>
```

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 `wasspi_wbs_UDMMetrics-sample.xml` file:

  ```
  wasspi_perl_su -S wasspi_ca -prod wbs -a -m TestUDM_1000 -i examplesServer
  ```

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

  ```
  wasspi_perl_su -S wasspi_ca -prod oas -a -m TestUDM_1001 -i examplesServer
  ```

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

  ```
  wasspi_perl_su -S wasspi_ca -prod wls -a -m TestUDM_1001 -i examplesServer
  ```

Index

We appreciate your feedback!

If an email client is configured on this system, by default an email window opens when you click on the bookmark “Comments”.

In case you do not have the email client configured, copy the information below to a web mail client, and send this email to **docfeedback@hp.com**

Product name:

Document title:

Version number:

Feedback: