# Oracle® Enterprise Manager

## Cloud Control Extensibility Programmer's Reference

13c Release 4

F23290-02

October 2020

ORACLE®

# Contents

# 3   Creating Target Metadata Files

# 4  Plug-in Builder

# 5    Adding Information Publisher Reports

# 6    Developing BI Publisher Reports

# 7  Collecting Target Configuration Data

# 8  Adding Job Types

## 9    Defining a Management User Interface

# 10    Customizing Incident Manager

# 11    Using Derived Associations

## 12    Defining Target Discovery

## 13   Adding Compliance Standards

## 14   Validating, Packaging, and Deploying the Plug-in

## 15     Defining Software Library Metadata

## 16     Defining Credentials

# 17    Defining a Chargeback Entity Type

# 18    Monitoring Using Web Services and JMX

# 19   Using Receivelets

# 20   Using Fetchlets

# 21 Enterprise Manager DTD

## A    Out-of-Box Associations

## B    Plug-in Technical Checklist

## C    Metric Unit Standardization

## Index

# List of Figures

# List of Tables

# Preface

This document covers Oracle Enterprise Manager framework extensibility and related reference information.

## Audience

This document is intended for plug-in developers that want to extend Oracle Enterprise Manager to support the ability to manage custom target types or extend the manageability of out-of-box target types.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

## Related Resources

For detailed information on Oracle Enterprise Manager, see the documentation available in Oracle Help Center:

`https://docs.oracle.com/en/enterprise-manager/`

## Conventions

The following text conventions are used in this document:

| Convention | Meaning |
| --- | --- |
| **boldface** | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |
| *italic* | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| `monospace` | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

# 1

# Getting Started with Plug-in Development

This chapter contains the following sections:

- About the Plug-in Creation Process
- About the Extensibility Development Kit (EDK)
- Installing the Extensibility Development Kit (EDK)
- Designing the Plug-in
- Creating a Basic Plug-in
- Creating an Intermediate Plug-in
- Creating an Advanced Plug-in

## About the Plug-in Creation Process

Creating a plug-in involves the following steps, all of which are covered in this book:

1. Designing your plug-in.

2. Developing the plug-in, which includes creating the requisite metadata files that enable the plug-in functionality.

3. Staging the plug-in.

4. Validating the plug-in.

5. Packaging the plug-in as an archive (.opar) file.

6. Importing the plug-in into Enterprise Manager Cloud Control.

7. Deploying the plug-in to Oracle Management Service.

8. Adding a target from your environment to initiate target monitoring. The plug-in files required by the Management Agent to monitor the target will be pushed to the Agent at this time.

9. Testing the plug-in to verify that it is behaving as expected.

As you continue to modify your plug-in metadata, you can upload your new metadata files to Enterprise Manager without re-deploying the plug-in archive using the Metadata Registration Service. See Updating Deployed Metadata Files Using the Metadata Registration Service (MRS) for instructions on using this service.

In addition, to keep track of each updated version of your plug-in, you should incrementally update the plug-in version as follows in the following plug-in metadata files:

- In the `PluginVersion` attribute in the plugin.xml file that describes the plug-in to Oracle Management Service the plug-in is deployed to. See Creating the plugin.xml File.

- In the `Version` attribute in the plugin_registry.xml file that describes the plug-in to Management Agents the plug-in is deployed to. See Creating the plugin_registry.xml File.

# About the Extensibility Development Kit (EDK)

A key component of the Enterprise Manager Cloud Control architecture is the Extensibility framework. To enable Oracle partners to extend the Enterprise Manager platform, an Extensibility Development Kit (EDK) is provided with the product.

The EDK is a collection of tools, utilities, and documentation, including:

- Enterprise Manager Extensibility documentation: Provide general guidelines for programming Enterprise Manager plug-ins

- Reference Implementation: Provides a reference code implementation, code snippets, and so on for various Enterprise Manager features

- Build time tools to verify EDK conformance: A tool that you can use to validate and report any violations, with respect to Enterprise Manager Extensibility guidelines

- Packaging Tool: A tool to package the plug-in components tool (`empdk`)

- Verification Tool: A tool to validate plug-in code components and to report violations (if any).

Enterprise Manager EDK includes a command line utility called `empdk`. Use this utility to package or validate a plug-in archive. For information associated with the `empdk` commands and their options, see Validating the Plug-in.

After you download the EDK, unzip it on your local system, and change your current directory to the location where you unpacked the EDK. The EDK contains reference documentation and guides to help you with plug-in development as well as the API reference you might need to integrate while developing plug-ins.

For information about downloading the EDK, see Installing the Extensibility Development Kit (EDK).

## Contents of the EDK

The EDK archive contains the following directories:

- /bin

  Contains the `empdk` utility, which you use to:

  – Validate the structure of your plug-in

  – Package your plug-in

  – Convert the metadata for existing (pre-Cloud Control 12) plug-ins to the new metadata formats

- /doc

  Contains the Oracle Enterprise Manager Extensibility Programmer's Guide and Programmer's Reference, as well as the EDK API Reference documentation, including documentation on Management Views. Review overview.html for links to the documentation provided.

You can also access the EDK API Reference documentation directly through its index page (sdk_api_ref.html).

- /lib

  Contains internal libraries used by the EDK.

- /oui

  Contains internal libraries used by the EDK.

- /samples

  Contains a complete reference implementation of a plug-in, packaged as demo_hostsample.zip. The sample metadata files included should be used as examples of the files referenced throughout the EDK documentation.

  View the README packaged with the archive for instructions on building, deploying, and using the sample plug-in.

Other utilities referenced in this documentation, including EM CLI and EM CTL, are installed with Enterprise Manager and are typically deployed to the Oracle Management Service (OMS) host.

# Installing the Extensibility Development Kit (EDK)

Before you begin developing plug-ins, install the Extensibility Development Kit (EDK).

> **Note:**
>
> Before installing the EDK, you must have the following:
>
> - Latest version of the EDK ZIP archive from the Self Update console. (To access the Self Update console, from the Cloud Control console, select **Setup**, then **Extensibility**, and then **Self Update**.)
> - Java version 1.7.0_51 or later
> - Local system running Solaris, Linux, HP-UX, AIX, or Windows with New Technology File System (NTFS)

To install the EDK:

1. Download the EDK ZIP archive to your local system using one of the following options:

   - Enterprise Manager Cloud Control

     a. Log in to Enterprise Manager Cloud Control.

     b. From the **Setup** menu, select **Extensibility**, then select **Development Kit**.

        The Extensibility Development Kit (EDK) page appears.

     c. Under Installing the EDK, select **Download the Extensibility Development Kit to your workstation**.

     d. Save 13.1.0.0.0_edk_partner.zip to your local system.

   - Enterprise Manager Command Line Utility (EM CLI)

> **✎ Note:**
>
> For information about setting up EM CLI, see Setting Up the EM CLI Utility.

Open a command prompt and run the following command:

```
emcli get_ext_dev_kit
```

This command downloads the EDK zip archive to the same directory from where you ran the command and does not require any parameters.

2. Set your JAVA_HOME environment variable and ensure that it is part of your PATH. For example:

```
setenv JAVA_HOME /usr/local/packages/j2sdk1.7.0_51
```

```
setenv PATH $JAVA_HOME/bin:$PATH
```

3. Unpackage the downloaded EDK ZIP archive to a directory on your local system. For example:

```
Unzip 13.1.0.0.0_edk_partner.zip
```

This command creates the following directories under the directory (*release_edk_partner*) where you unpackaged the EDK ZIP archive:

```
release_edk_partner
              |
             bin
             doc
             lib
             oui
             samples
             README
```

For more information about the directory contents, see Contents of the EDK.

# Designing the Plug-in

Before creating your plug-in, you must first determine what information needs to be collected to monitor and manage the target type. This involves:

- Identifying performance and configuration metrics that should be collected.

- Determining how often each metric should be collected. Oracle recommends that the collection frequency for any metric should not be less than once every five minutes.

- Based on customer-specific operational practices, specifying default warnings and critical thresholds on these metrics. Whenever a threshold is crossed, Enterprise Manager generates an alert, informing administrators of potential problems.

# Creating a Basic Plug-in

Ideally, begin by creating a basic monitoring plug-in that includes the basic required metadata:

- The target type definition file, which defines:
    - A required "Response" metric group, which includes a status metric and a performance metric.
    - Credentials needed to authenticate with the target.

    For more information, see Creating Target Metadata Files .

- A default collection file defining the frequency at which metrics and configuration data will be collected. For more information, see Creating the Default Collection File.

- Developing Oracle Business Intelligence Publisher (BI Publisher) reports to display collected target data.

    For more information, see Developing BI Publisher Reports .

Once created, you will validate and package your plug-in. See Validating, Packaging, and Deploying the Plug-in for instructions.

As the final step, deploy your plug-in to Enterprise Manager Cloud Control and begin testing to ensure that data for the Response metric is being returned.

> **Note:**
>
> The EDK provides a sample host plug-in. This contains examples of all features mentioned previously. For more information, see the README bundled with the sample plug-in.

# Creating an Intermediate Plug-in

When you have created a basic plug-in, you might want to enhance the plug-in's capabilities by adding more complex functionality.

- Add more complex metrics.

    For more information, see Defining Advanced Metrics.

- Provide the ability to collect configuration data for the target, which is used to create a "snapshot" of the target's configuration at a specific point in time.

    For more information, see Collecting Target Configuration Data.

- Create a metadata-based metadata custom user interface, which will essentially add a custom target home page for displaying target metrics to Enterprise Manager.

    For more information, see Defining a Management User Interface .

- Defining target associations, which can be used to create topology models of the targets managed by the plug-in.

    For more information, see Using Derived Associations.

- Define a job type that executes specific tasks specific to the target type.

    For more information, see Adding Job Types .

# Creating an Advanced Plug-in

When you have successfully created and validated your basic or intermediate plug-in, you might want to enhance it with advanced features. Among the enhancements to consider:

- Adding the ability to automatically discover newly-added instances of the target type managed by the plug-in.

  For more information, see Defining Target Discovery .

- Building a Flex-based custom management user interface. This page will be accessed with other target home pages through the Cloud Control console.

  For more information, see Defining a Management User Interface .

- Define compliance standards and monitoring rules specific to the target type.

  For more information, see Adding Compliance Standards.

# 2

# Defining the Plug-in

This chapter describes the plug-in metadata that you must define to create a plug-in It contains the following sections:

- Introduction to Defining the Plug-in
- About Plug-in Metadata
- Creating Plug-in Definition Files
- Validating Plug-in Definition Files
- Adding Log Viewer Support to Your Plug-in
- Defining Plug-ins for Upgrade
- Deprecating a Plug-in

## Introduction to Defining the Plug-in

As a plug-in developer, you are responsible for the following steps within the plug-in definition process:

1. Define the plug-in identifier (ID).

   For more information, see Defining the Plug-in ID.

2. Define the plug-in version.

   For more information, see Defining the Plug-in Version.

3. Create the plug-in definition files:

   a. Create the plugin.xml file.

      The plugin.xml file provides the metadata describing the plug-in.

      For more information, see Creating the plugin.xml File.

   b. Create the plugin_registry.xml file.

      The plugin_registry.xml file provides the metadata required by the Management Agent to which the plug-in will be deployed.

      For more information, see Creating the plugin_registry.xml File.

4. Package the plug-in definition files in the plug-in staging directory (*plugin_stage*):

   - `plugin_stage/plugin.xml`
   - `plugin_stage/agent/plugin_registry.xml`

   For more information, see Validating, Packaging, and Deploying the Plug-in .

5. Validate the plug-in definition files.

   For more information, see Validating Plug-in Definition Files

# About Plug-in Metadata

A basic plug-in requires metadata for the plug-in itself, including information such as the name and version that is used by Oracle Management Service and Management Agents, definition of a metric indicating whether the monitored target is up, and definition of the frequency at which metric data should be collected.

## Defining the Plug-in ID

Plug-ins are identified by a unique plug-in identifier (ID). The plug-in ID has three parts:

- Vendor ID (8 chars). For example: `test`

- Product ID (8 chars). For example: `switch`

- Plug-in Tag (4 chars). For example: `xkey`

> ✏️ **Note:**
>
> - The Vendor ID, Product ID, and Plug-in Tag *must not* begin with a number or include any special characters. All these parts must contain alphanumeric characters only.
>
> - The Plug-in Tag *must* begin with a lower-case x and cannot exceed 4 characters. All characters must be lower case.
>
> - If you are planning to define more than one plug-in, then make sure that the Plug-in Tag for each plug-in is distinct and unique.
>
> - If you want to maintain the previous names of existing plug-ins, then you must use the `AgentSideCompatibility` element. Otherwise, plug-in validation will fail. For information about plug-in validation, see Validating, Packaging, and Deploying the Plug-in . For information about the `AgentSideCompatibility` element, see Table 2-1.

The Plug-in ID created from the previous example would be:

```
test.switch.xkey
```

> ✏️ **Note:**
>
> The plug-in ID must be unique across Enterprise Manager.

## Defining the Plug-in Version

Each plug-in must be assigned a version. The plug-in versioning syntax is as follows:

```
a.b.c.d.e
```

- $a.b$ = The version of the Enterprise Manager Extensibility Development Kit (EDK) used for development (13.1, 13.2, and so on).

- $c$ = The developer-assigned plug-in version. This value must be incremented with each plug-in release on the same Enterprise Manager Cloud Control release.

- $d$ = Indicates whether the plug-in is a beta version or a production version. 0 indicates beta and 1 or later indicates production.

- $e$ = For future use. The default value is 0.

Putting it all together, the following example shows the first version of a plug-in created for Enterprise Manager Cloud Control 13*c*:

```
13.1.1.1.0
```

> **Note:**
>
> Oracle recommends that you update the plug-in version incrementally as you create and deploy each iteration of your plug-in. For example, 13.1.1.1.0, 13.1.2.1.0, 13.1.3.1.0, and so on.

# Creating Plug-in Definition Files

The following two metadata files are required for all plug-ins deployed to Enterprise Manager Cloud Control 13*c*.

- plugin.xml

  This file is used during plug-in deployment. It contains properties that identify the plug-in, such as name and version, and declares the set of target types that will be added to Enterprise Manager Cloud Control.

- plugin-registry.xml

  This file declares those components included in the plug-in that are to be deployed to the Management Agent.

## Creating the plugin.xml File

The plugin.xml file provides the metadata describing the plug-in.

The following sections describe the required and some of the optional tags that you can include in the plugin.xml file.

This example provides a sample plugin.xml for a plug-in.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/plugin_
metadata plugin_metadata.xsd"
        xmlns="http://www.oracle.com/EnterpriseGridControl/plugin_metadata">

 <PluginId vendorId="test" productId="demo" pluginTag="xkey"/>

  <PluginVersion value="13.1.1.1.0"/>
```

```
<ShortDescription>Test plugin for the Test Demo Plug-in.</ShortDescription>
<Readme><![CDATA[Brief details about the Test Demo plug-in]]></Readme>
<!--
<AgentSideCompatibility>
    <Version>Previous_Version</Version>
</AgentSideCompatibility>
-->

<TargetTypeList>
  <TargetType name="test_demo_xkey" isIncluded="TRUE">
     <VersionSupport>
      <SupportedVersion supportLevel="Basic" minVersion="9.2.0.1"
                        maxVersion="9.8.0.0.0"/>
    </VersionSupport>
  </TargetType>
</TargetTypeList>

<PluginDependencies>
 <DependentPlugin pluginDependencyType="RunTime">
  <DepPluginId vendorId="test" productId="switch" pluginTag="xyz1"/>
  <BaseVersion version="11.2.0.1.0"/>
 </DependentPlugin>
 <DependentPlugin pluginDependencyType="RunTime">
  <DepPluginId vendorId="test" productId="switch" pluginTag="xyz2"/>
  <BaseVersion version="11.2.0.1.0"></BaseVersion>
 </DependentPlugin>  </PluginDependencies>
 <PluginAttributes Type="MP" Category="Databases"/>
</Plugin>
```

# Overview of plugin.xml Elements

Table 2-1 describes the key elements included within the plugin.xml files.

**Table 2-1    Key Elements Within the plugin.xml File**

| Element | Required | Description |
|---|---|---|
| Plugin | Y | The root element for the file. |
| PluginID | Y | The unique identifier assigned to the plug-in. |
| | | For more information, see Defining the Plug-in ID. |
| PluginVersion | Y | The plug-in version. |
| | | For more information, see Defining the Plug-in Version. |
| PluginOMSOSAruId | N | The operating system (OS) ID for the Oracle Management Service to which the plug-in will be deployed. Usually, this element is set to 2000 (generic). |
| | | For more information, see Certifying Plug-ins. |
| Readme | Y | Provides information about the plug-in that is displayed on the **Plug-ins** page of the Cloud Control console. |
| | | To access the **Plug-ins** page, from the **Setup** menu, select **Extensibility**, then **Self Update**, and then **Plug-ins**. |

**Table 2-1    (Cont.) Key Elements Within the plugin.xml File**

| Element | Required | Description |
| --- | --- | --- |
| PluginAttributes | N | Defines plug-in attributes such as plug-in type, display name, category, and so on. |
| | | The default Plug-in Type for metadata plug-ins is "MP". The default Category is "Others". |
| | | Valid Type values: |
| | | • MP |
| | | Metadata plug-in with default UI |
| | | • MPP |
| | | Metadata plug-in with custom UI |
| | | Valid Category values: |
| | | • Applications |
| | | • Databases |
| | | • Middleware |
| | | • Cloud |
| | | • Engineered Systems |
| | | • Servers, Storage and Network |
| | | • Others |
| | | **Note**: Oracle recommends that you use a specific category value rather than **Others**. |
| TargetTypeList | N | Contains one or more TargetType elements, each specifying the target type name packaged with the plug-in. |
| | | For information about the target type metadata file, see Creating the Target Type Metadata File. |
| | | Each TargetType element can also include a VersionSupport element identifying supported or non-supported versions of the target type. |
| PluginDependencies | N | Describes any dependencies that exist for the plug-in. Dependencies can be described as the following: |
| | | • RunTimeMandatory: This dependency indicates that the plug-in that the current plug-in is dependent on, *must* exist before deploying the current plug-in. For example, Plug-in A cannot be deployed if the plug-in on which it depends does not exist. Plug-in A uses a feature from plug-in B at runtime but this feature will break if plug-in B is missing. |
| | | • RunTime: This dependency indicates that the deployment of the current plug-in with feature dependencies can go ahead without the plug-in that it is dependent on. If the plug-in that the current plug-in is dependent on comes into the environment later, then dependent features will be enabled. Along with dependencies, you can also describe the prerequisites. Currently the supported prerequisite is of type bug. |
| | | • CompileTime: This dependency indicates that the dependent plug-in should exist before deployment of the current plug-in, that is, if the current plug-in explicitly consumes an API from a dependent plug-in and has a build-time dependency. |

**Table 2-1 (Cont.) Key Elements Within the plugin.xml File**

| Element | Required | Description |
|---|---|---|
| `AgentSideCompatibility` | N | Identifies the previous plug-in versions with which the current plug-in is compatible. |
| | | By specifying this element, you indicate explicitly that the previous plug-in metadata on the Management Agent side is compatible with the new version of plug-in metadata on the OMS side. That is, after upgrading the previous plug-in, you can upload data to the new version on OMS without breaking any features, such as metrics, thresholds or configuration collections. |
| | | If you have a previous version of a plug-in that is not compatible with the new version of the plug-in, then you can use this element to list the compatible versions only. For example, if version 12.1.0.2.0 is not compatible with 13.1.1.1.0 (new plug-in version), then you can list 12.1.0.3.0 and 12.1.0.4.0 to indicate that *only* 12.1.0.3.0 and 12.1.0.4.0 plug-ins are compatible with the new 13.1.1.1.0 plug-in. |

## Certifying Plug-ins

> **Note:**
>
> All metadata plug-ins must be generic on the OMS side and are implicitly certified on all platforms. However, the plug-in can specify the OS certification for the Management Agent.

Because Enterprise Manager is released on a number of OS platforms, you must consider how your plug-in will behave on different OS platforms. The plugin.xml file contains elements and attributes that support a certification mechanism.

In cases, where the plug-in is applicable to only a subset of OS platforms, you can use the tags defined in Table 2-2. If you do not specify any information in the `<Certification>` section, the plug-in is assumed certified on all platforms.

**Table 2-2 Certification Tags**

| Tag | Description |
|---|---|
| `Component type` | Specifies the plug-in component.<br>Valid values:<br>• Agent: Management Agent component<br>• Discovery: Discovery component |

**Table 2-2    (Cont.) Certification Tags**

| Tag | Description |
| --- | --- |
| `PortARUId value` | Specifies the ARU ID for the OS or platform. |
| | Valid values: |
| | • 46: Linux x86 (32-bit) |
| | • 212: AIX 5L and 6.1 (64-bit) |
| | • 226: Linux x86-64 (64-bit) |
| | • 23: Solaris Sparc (64-bit) |
| | • 267: Solaris x86-64 (64-bit) |
| | • 233: Microsoft Windows x86-64 (64-bit) |

The following example indicates that the plug-in is designed to work on Linux 32 and Linux 64 platforms only. If you do not specify a certified port, then by default your plug-in is certified on all operating systems and platforms. But if you specify at least one `PortARUId` element, then that component is certified on those specified platforms *only*.

> **Note:**
>
> The Management Agent and Discovery components must have the same value

**Example: Certifying Generic Plug-ins**

```
<Certification>
  <Component type="Agent">
      <CertifiedPorts>
            <PortARUId value="46" />
            <PortARUId value="226" />
      </CertifiedPorts>
  </Component>
  <Component type="Discovery">
      <CertifiedPorts>
            <PortARUId value="46" />
            <PortARUId value="226" />
      </CertifiedPorts>
  </Component>
</Certification>
```

## Creating the plugin_registry.xml File

The plugin_registry.xml file provides the metadata required by the Management Agent that the plug-in will be deployed to. It is packaged in the /agent directory within the plug-in archive and is deployed to the Management Agent that will monitor a target.

The following example provides a sample plugin_registry.xml file. The `TargetTypes` element contains a reference to the target type metadata file location in the plug-in archive. The location is relative to the *plugin_stage* directory root, that is, starting from the Management Agent subdirectory or the same location where the plugin_registry.xml file is located.

Similarly, the `TargetCollections` element contains a reference to the plug-in's default collection metadata file, which is also packaged with the plug-in.

**Example: Sample plugin_registry.xml File**

```
<?xml version="1.0"?>
<PlugIn ID="test.demo.xkey" Description="Demo Sample Host Plugin"
Version="13.1.1.1.0" HotPluggable="false"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/plugin
plugin.xsd">
  <TargetTypes>
        <FileLocation>metadata/test_switch_key.xml</FileLocation>  </TargetTypes>
  <TargetCollections>
       <FileLocation>default_collection/test_switch_key_collection.xml</
FileLocation>
  </TargetCollections>
  <PlugInLibrary>
    <FileLocation>archives/em-as-fmw-fetchlet.jar</FileLocation>
    <FetchletRegistration>
      <Fetchlet ID="DMS" ExecutionClass="oracle.sysman.as.fetchlets.DMSFetchlet"
Version="" Description="" Adapter=""/>
    </FetchletRegistration>
    <AdditionalClassPath>
      <FileLocation>archives/dms.jar</FileLocation>
    </AdditionalClassPath>
  </PlugInLibrary>
</PlugIn>
```

## Overview of plugin_registry.xml Elements

Table 2-3 describes the key elements included within the file.

**Table 2-3    Key Elements Within the plugin_registry.xml File**

| Element | Required | Description |
|---|---|---|
| Plugin | Y | The root element for the file. It includes the following attributes:<br>• ID: Required. The unique identifier assigned to the plug-in.<br>  For more information, see Defining the Plug-in ID.<br>• Description: Optional. A title describing the plug-in.<br>• Version: Required. The plug-in version.<br>  For more information, see Defining the Plug-in Version. |
| TargetTypes | N | Contains one or more `FileLocation` elements, each specifying the path and file name for a target type metadata file packaged with the plug-in.<br>For information about target type files, see Creating the Target Type Metadata File. |
| TargetCollections | N | Contains one or more `FileLocation` elements, each specifying the default collection for a target type.<br>For information about this file, see Creating the Default Collection File. |

**Table 2-3    (Cont.) Key Elements Within the plugin_registry.xml File**

| Element | Required | Description |
|---|---|---|
| PlugInLibrary | N | Lists the different types of artifacts (fetchlets, receivelets, and so on) packaged in the plug-in.<br><br>The `PlugInLibrary` element includes the following subelements:<br>• `FileLocation`: Mandatory. Defines the location of the JAR containing the implementations of the following listed fetchlets.<br>• `FetchletRegistration`: Optional. Creates an entry that maps a fetchlet id (**DMS** in the Sample plugin_registry.xml File example) to the class that contains the implementation of the fetchlet interface.<br>• `ReceiveletRegistration`: Optional. Creates an entry that maps a receivelet id to the class that contains the implementation of the receivelet interface<br>• `AdditionalClassPath`: Optional. Specifies additional JAR files to be loaded by the plug-in for a specific library. |
| AdditionalClassPath | N | Specifies additional JAR files to be loaded by the plug-in that are shared by all the plug-in libraries |

# Validating Plug-in Definition Files

To verify that your plugin.xml and plugin_registry.xml files are defined correctly, enter the following command from the /bin directory of the EDK:

```
empdk validate_plugin -stage_dir plugin_stage
```

In the preceding command, *plugin_stage* represents the plug-in staging directory.

For information about the EDK, see About the Extensibility Development Kit (EDK) and for more information about the `empdk` command and its usage, see Validating the Plug-in.

# Adding Log Viewer Support to Your Plug-in

Beginning with Enterprise Manager Cloud Control Release 12*c* (12.1.0.3), you can enable log files for your deployed plug-in to be viewable with Cloud Control's Log Viewer. To access this component, from the Cloud Control **Enterprise** menu, select **Monitoring**, then **Logs**.

Follow these steps to enable this feature:

1. Create the log viewer registration XML file for your plug-in. The DTD for this XML file is:

   oracle/sysman/emSDK/logmgmt/registration/LogMgmtTargetTypeRegistration.xsd

2. Package this file in oms/metadata/logmgmt/ within the plug-in directory structure.

The following example provides an example of a log viewer registration file.

**Example: Sample Log Viewer Registration File**

```
<LogMgmtUITargetConfig TARGET_TYPE="%your targe type%">
  <LogViewerImpl CLASS_NAME="oracle.sysman.emas.model.logmgmt.MASLogViewer"/>
```

```
    <VersionProperties VALID_VERSIONS="11" MIN_META_VER="11.00000"VERSION_
      CATEGORY_PROP_WILDCARD_CHAR="*"/>
</LogMgmtUITargetConfig>
```

# Defining Plug-ins for Upgrade

During plug-in development, if you are planning a subsequent version of your plug-in, you must ensure that the plug-in can be upgraded, that is, you can deploy a new version without having to remove an older version of the plug-in.

To ensure that your plug-in can be upgraded:

1.  In the new plug-in.xml file, include the `AgentSideCompatibility` tag explicitly specifying the compatible previous versions of the plug-in.

    For more information about the `AgentSideCompatibility` tag, see Table 2-1.

    > **Note:**
    >
    > - Oracle recommends that you include at least the two previous versions.
    >
    > - If the previous plug-in metadata is not compatible with the new version, then you might see metrics and collection errors after upgrading. Oracle recommends that you list the compatible versions of the plug-in under the `AgentSideCompatibility` element to avoid issues with upgrading your plug-in.

2.  When you specify previous plug-in versions under the `AgentSideCompatibility` element, you must bundle the OPARs of these plug-in versions in the plug-in staging directory.

    Under the plug-in staging directory (*plugin_stage*), create a released_plugins directory and place the previously released plug-in OPAR archive (for example, 12.1.0.2.0_test.demo.xkey_2000_0.opar) in this directory.

    For more information about the plug-in staging directory, see Staging the Plug-in.

3.  Perform upgrade testing from all previous versions of the plug-in to ensure that all features are working correctly such as:

    - Verify metric collections

    - Verify configuration collections

    - Check that there are no metric collection errors due to the upgrade

    - Check that updated metric thresholds and templates are picked up as expected

    - Check that updated job metadata is picked up as expected

# Deprecating a Plug-in

To mark your plug-in for deprecation, add the following line to your plugin.xml file:

```
<Deprecated />
```

For example:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<Plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/
plugin_        metadata plugin_metadata.xsd"
        xmlns="http://www.oracle.com/EnterpriseGridControl/plugin_metadata">

  <PluginId vendorId="test" productId="demo" pluginTag="xkey"/>

  <PluginVersion value="12.1.0.1.0"/>

  <Deprecated/>

  <ShortDescription>Test plugin for the Test Demo Plug-in.</ShortDescription>
  <Readme><![CDATA[Brief details about the Test Demo plug-in]]></Readme>
  .
  .
  .
```

> **Note:**
>
> When a plug-in is marked as deprecated, it continues to provide the same level of support until it is obsolete in the next major release.

# 3

# Creating Target Metadata Files

This chapter provides the steps involved in the target metadata file creation process. It contains the following sections:

- Introduction to Creating Target Metadata Files
- Overview of Target Definition Files
- Creating the Target Type Metadata File
- Defining Metrics to Collect from the Target
- Creating the Default Collection File
- Guidelines for Creating Target Metadata
- Localizing Target Metadata
- Checking a New Target Type
- Testing Your Target Type Definitions
- Validating Your Metadata XML
- Troubleshooting the Target Creation Process

## Introduction to Creating Target Metadata Files

As a plug-in developer, you are responsible for the following steps within the target metadata files creation process:

1. Create the target definition file.

   The target type metadata file tells the Management Agent what data to retrieve and how to obtain that data for this particular target type.

   For more information, see Creating the Target Type Metadata File.

2. Define metrics to collect from the target.

   A metric refers to a specific piece of data collected from the target. A set of related metrics collectively comprise a metric group.

   For more information, see Defining Metrics to Collect from the Target.

3. Define target configuration data to collect.

   You can collect configuration data for a target and save it in the Management Repository as a *snapshot* representing the target's configuration at a specific point in time. Each configuration snapshot is associated with a specific target instance.

   For more information, see Collecting Target Configuration Data.

4. Create the default collection file.

   The default collection file defines the metric data to be collected from targets and uploaded to the Management Repository along with information such as the collection schedule.

For more information, see Creating the Default Collection File.

5. Package the various definition files in the plug-in staging directory (*plugin_stage*):

- Target type metadata file

    — *plugin_stage*/oms/metadata/targetType/*target_type*.xml

    — *plugin_stage*/agent/metadata/*target_type*.xml

    > **Note:**
    >
    > An identical copy of this file must be placed in both the /oms and /agent directories.

- Default collection file

    — *plugin_stage*/oms/metadata/default_collection/*target_type*.xml

    — *plugin_stage*/agent/default_collection/*target_type*.xml

    > **Note:**
    >
    > An identical copy of this file must be placed in both the /oms and /agent directories.

- Configuration metadata file

    *plugin_stage*/oms/metadata/snapshotlive/*target-type*_ecmdef.xml

    For more information, see Validating, Packaging, and Deploying the Plug-in .

# Overview of Target Definition Files

Two XML metadata files are required to define the target type that your plug-in will enable Enterprise Manager Cloud Control to monitor and manage:

- Target type metadata file

  The definition of a target type primarily consists of the metrics you want the Management Agent to collect for the target. The file contains a list of all metrics that will be collected for the target type, along with specifics on how to compute each metric.

  For more information, see Creating the Target Type Metadata File.

- Default collection file

  This file defines the interval at which metric data will be collected or received from the target. You can specify optional alert thresholds and optional corresponding alerts messages for each metric. Cloud Control users can override the default collection intervals, but the default values must be provided in this file.

  For more information, see Creating the Default Collection File.

This chapter also describes the metadata definitions required to collect configuration data for plug-in targets. This is an advanced feature but can be useful for many plug-ins. For more information, see About the Configuration Definition Files.

The following sections provide the summary of creating the target type and default collection metadata files, as well as an overview of target configuration data collection.

# Creating the Target Type Metadata File

The target type metadata file tells the Oracle Management Agent what data to retrieve and how to obtain that data for this particular target type.

At the highest definition level, the target type metadata file is composed of four key XML elements as described in Table 3-1.

**Table 3-1    Key Elements of the Target Type Metadata File**

| Element | Description |
| --- | --- |
| `TargetMetadata` | Specifies information about the plug-in, such as name and version. |
| `Metric` | Defines a metric group, which in turn contains one or more metrics that each define a specific piece of data collected from the target. |
| `InstanceProperties` | Defines properties that are populated when a target instance is created. |
| `CredentialTypes/CredentialSets` | Specifies credentials required to by the plug-in authenticate with a target instance. |

Enterprise Manager ships with predefined target type metadata files that cover the most common target types. In situations where the predefined target metadata files do not fit the types of targets you want to monitor, you can either:

- Define a new target type by creating a target type metadata file
- Use one of the predefined metadata files as a template for defining a new target type, and then repackage the files as a new plug-in

This section briefly introduces the structure of the target type metadata file. A complete example of a target type metadata file is provided with the EDK:

*edk*/samples/plugins/SampleHost/oms/metadata/targetType/demo_hostsample.xml

In the preceding directory path, *edk* represents the directory where you expanded the EDK archive. For information about the EDK archive, see About the Extensibility Development Kit (EDK).

For additional information about creating target type metadata files, Guidelines for Creating Target Metadata.

## Creating a Basic Target Type Metadata File

The following example shows the minimum required information that a target type file must contain.

**Example: Target Type File**

```
<TargetMetadata META_VER="2.0" TYPE="demo_hostsample">
  <Display>
    <Label NLSID="hs_displayname">Demo Plugin Sample Host</Label>
  </Display>
```

```
       <Metric NAME="Response" TYPE="TABLE">
         <Display>
           <Label NLSID="hs_response_displayname">Response</Label>
         </Display>
         <TableDescriptor>
           <ColumnDescriptor NAME="Status" TYPE="NUMBER">
             <Display>
               <Label NLSID="hs_response_status">Status (up/down)</Label>
             </Display>
           </ColumnDescriptor>
         </TableDescriptor>
         <QueryDescriptor FETCHLET_ID="OSLineToken">
           <Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property>
           <Property NAME="fake" SCOPE="INSTANCE"
                 OPTIONAL="TRUE">USE_FAKE_DATA</Property>
           <Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>
           <Property NAME="script" SCOPE="GLOBAL">%scriptsDir%/emx/demo_hostsample/
datacollector.pl --collect Response --fake %fake%</Property>
           <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
           <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
         </QueryDescriptor>
       </Metric>
       <InstanceProperties>
         <InstanceProperty NAME="SAMPLE_DATA" CREDENTIAL="FALSE" OPTIONAL="TRUE">
           <Display>
             <Label NLSID="EMPLOYEE_ID_iprop">Employee ID</Label>
           </Display>
         </InstanceProperty>
      </InstanceProperties>
</TargetMetadata>
```

The following sections provide information about the XML definitions shown in the previous example.

# Naming the Target Type Metadata File

Oracle recommends that users adding new target types adhere to Enterprise Manager naming conventions. This naming convention allows for file naming consistency in environments where similar products from multiple vendors are used. The target naming convention follows the form *vendorID_productID_PluginTag*.

For example:

```
test_demo_targetType.xml
```

# Defining the Target Type Metadata

The first lines after the header of the target definition file identify the target type. The following excerpt defines the metadata version (META_VER="2.0") and target type (TYPE="test_demo_targetType").

```
<TargetMetadata META_VER="2.0" TYPE="test_demo_targetType">
```

The TYPE attribute and the META_VER attribute of the TargetMetadata element must match the TYPE attribute and the META_VER attribute of the TargetCollection element of the default collection metadata file. For more information about the default collection metadata file, see Creating the Default Collection File.

Metadata versioning allows different versions of the same target type metadata to exist concurrently within the managed environment, although only one metadata version is allowed per Management Agent. You should update the metadata version each time you update the target metadata file.

The syntax for the meta value is MajorNumber.MinorNumber. The value for MinorNumber can be either one or two digits. It is important that you choose to use either one or two digits for MinorNumber, and continue to use that syntax throughout the plug-in's lifecycle.

A one-digit MinorNumber can be updated up to nine times, after which the MajorNumber value must be increased. For example:

```
1.0, 1.1, 1.2, ...,1.9, 2.0, 2.1
```

A two-digit MinorNumber can be revised up to 99 times. For example:

```
1.01, 1.02, 1.09, ..., 1.10, 1.11, ..., 1.99, 2.01
```

Note that you cannot combine single-digit and two-digit formats. For example, increasing the version from 1.9 to 1.10 is not valid.

If you modify the metadata for one or more metrics, or change credentials, or update the target property file during plug-in development, be sure to increment the value of the META_VER attribute of the `TargetMetadata` element to the next digit value. For example, if the current version is "1.0", set the value to "1.1" if you modify the metric metadata:

```
<TargetMetadata META_VER="1.1" TYPE="demo_hostsample">
```

Continue to increment the version each time you modify your metrics. Otherwise, NullPointerException errors may occur if the plug-in is deployed to the same development or test installation that the previous version was deployed to.

Once plug-in development is complete, the META_VER value can safely be set to the actual production version.

## Defining Target Credentials

In most cases, the plug-in will be required to authenticate with each target instance that it will collect data for, or run jobs against. Credential types and credential sets needed to enable authentication are defined in the `CredentialInfo` element within the target type metadata file.

For more information, see Defining Credentials.

Credentials information for the target includes and defines the credentials fields (referred to as columns) and the credentials sets specific to the target type. Enterprise Manager's security framework provides facilities for managing these credentials and using them when performing various management functions.

## Defining Type Properties

The extensibility framework uses type properties to internally categorize the target type for framework processing. They are not visible to the end user. Corresponding subsystems use the type properties to enable features for the target type or to perform appropriate validation checks.

The value set at the type property level applies to all targets of that type and across all metaversions, unlike instance properties which only apply to a specific target.

The following example specifies that the target type is a system class of target. The extensibility framework uses this setting to display the target on all system pages.

```
<TypeProperties>
    <TypeProperty PROPERTY_NAME="is_system" PROPERTY_VALUE="1"/>
</TypeProperties>
```

Table 3-2 provides a description of the available type properties.

**Table 3-2    Type Properties**

| Property Name | Description |
|---|---|
| is_system | Specifies the type as modelling a system type. You must set this value for all system types. <br><br> •    is_cluster: Specifies the type as modelling a cluster. Clusters are subsets of systems. <br> •    is_end_user: Set for systems constructed from user-chosen entities; these are subsets of systems. <br><br> **Note**: The property value is always set to 1 for all the is_*name* properties. |
| is_service | Specifies the type as modelling a service. <br><br> **Note**: The property value is always set to 1 for all the is_*name* properties. |
| is_aggregate | Enterprise Manager sets this value automatically. Do not modify. |
| is_group | Do not use |
| is_composite | Do not use |
| is_install | Set this value for an install home manageable entity (for example, Oracle home) |
| is_existence | Set this value for a discovered entity with an existence-only state, that is, an entity that is discovered but cannot be managed by Oracle yet. <br><br> Possible value: <br><br> •    1: Indicates a discovered entity with an existence-only state <br> **Note**: When the entity becomes a managed entity by Oracle, you must remove this entry from the target type metadata file and register the target type again. |
| priv_propagation_mode | This property is used for privilege propagation and specifies the privilege propagation mode. <br><br> Possible values: <br><br> •    0: No privilege propagation <br> •    1: Privilege propagation at instance level <br> •    2: All targets are privilege propagating |
| disallow_redundancy_group | Used by the redundancy group feature to disable redundancy groups for certain target types (which have disallow redundancy group set). Specifies whether redundancy group can contain this type as a member. <br><br> Possible value: <br><br> •    1: Do not allow redundancy |

**Table 3-2    (Cont.) Type Properties**

| Property Name | Description |
|---|---|
| member_target_type | Used by the redundancy group feature to lock the target type to the specified member_target_type. |
| TargetVersion | Specifies the name of the instance or dynamic property that represents the target version for the target type (for all target pages and plug-in certification).<br><br>**Note**: Oracle recommends including this property when you are defining target types. |

**Example: Defining Type Properties**

```
<TargetMetadata META_VER="1.1" TYPE="oracle_dbsys" CATEGORY_PROPERTIES=""
RESOURCE_BUNDLE_PACKAGE="oracle.sysman.db.rsc">
  <Display>
    <Label NLSID="oracle_dbsys_nlsid">Database System</Label>
  </Display>

  <TypeProperties>
    <TypeProperty PROPERTY_NAME="is_system" PROPERTY_VALUE="1"/>
    <TypeProperty PROPERTY_NAME="priv_propagation_mode" PROPERTY_VALUE="2"/>
  </TypeProperties>

  <MonitoringMode MEDIATOR="Repository"/>
</TargetMetadata>
```

# Defining Instance Properties

Instance properties are populated when a target instance is created. The InstanceProperties descriptor within the target type metadata file defines what properties an administrator must specify in the Enterprise Manager Cloud Control console when adding a new target instance of this particular target type.

Although the InstanceProperties section can be defined at various locations within the target type metadata file, Oracle recommends defining this section at the very end of the file for consistency. Instance properties defined in the target type metadata file are resolved to values specified for these instance properties in the target type metadata file.

Target instance properties are named values that can be used for computing the metrics of the target, or for display in the home page of the target. The list of target instance properties is specified in the metadata to allow data driven user interfaces to register targets, and for the Oracle Management Agent to validate that a target instance is complete.

# Static Instance Properties

Instance properties are populated when a target instance is created. In this example, the property is required (OPTIONAL="FALSE) and it is a credential property.

```
<InstanceProperties>
 <InstanceProperty NAME="password" OPTIONAL="FALSE" CREDENTIAL="TRUE">
  <Display>
    <Label NLSID="USER_PASSWORD">User Password</Label>
```

```
    </Display>
  </InstanceProperty>
</InstanceProperties>
```

# Dynamic Instance Properties

The values for dynamic instance properties are passed back by the Management Agent collecting data from the target instance. They are typically used within a `QueryDescriptor` to define properties passed to the fetchlet responsible for metric collection. For more information about the `QueryDescriptor` element, see Table 3-4. For more information about fetchlets, see Using Fetchlets .

The properties in the following example are described in Defining the Basic Response Metric Group.

```
<InstanceProperties>
 <DynamicProperties NAME="AruidInfo" FORMAT="ROW" OPT_PROP_LIST="ARUID">
  <QueryDescriptor FETCHLET_ID="OSLineToken">
    <Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property>
    <Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>
    <Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl</Property>
    <Property NAME="script" SCOPE="GLOBAL">%scriptsDir%/hostaruid.pl</Property>
    <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
    <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>  </QueryDescriptor>
 </DynamicProperties>
</InstanceProperties>
```

Using dynamic properties reduces the work involved in configuring a target by allowing certain properties to be computed rather than requiring the user to correctly specify their values (for example, the "Version" property of a database can be reliably computed given addressing information).

Dynamic properties are computed in the order that they are defined in the XML file so that later dynamic properties can use values from earlier dynamic properties in the XML file if required.

The Management Agent allows for the fact that the target must be up for the successful computation of these dynamic properties by recomputing the properties each time a target restart is detected; that is, each time the target status changes to **Up**.

> **Note:**
>
> Some properties can be computed without access to the target; therefore there is some support for computing dynamic properties when the target is down.
>
> To compute a dynamic property when the target is down, include the following attribute:
>
> `COMPUTE_WHEN_DOWN="TRUE"`

# Defining Metrics to Collect from the Target

Metrics are at the core of Cloud Control's target monitoring capabilities. When we speak of Cloud Control's ability to monitor and manage various targets, what we are really talking about is its ability to collect, process, and display target metrics.

A *metric* refers to a specific piece of data collected from the target.

Metrics can viewed as being of two basic types:

- Pull metrics

  In this model, the plug-in polls the target for metric data at the frequency specified in the default collections file. This is the most common type of metric utilized by plug-ins.

  A metric of this type requires the use of a *fetchlet*, a parametrized data access mechanism that takes arguments (for example, a script, a SQL statement, a target instance's properties) as input and returns formatted data.

  Predefined fetchlets are provided by Oracle for use with plug-ins. For a list of available fetchlets and information about their usage, see Using Fetchlets .

- Push metrics

  In this model, the plug-in receives notifications that are sent asynchronously from the target, without being requested. This type of metric requires a *receivelet*, which enables the plug-in to receive such notifications. As with fetchlets, predefined receivelets are provided by Oracle. For information about using receivelets, see Using Receivelets .

## Metric Definition Files

The target metadata must define each type of metric the plug-in will collect, how and when the metric data is to be collected, and what metric thresholds will trigger an incident to be raised within Cloud Control.

> **✐ Note:**
>
> All metrics that can be viewed from the Cloud Control UI *must* have a proper display label and NLSID.

The metadata for metric groups and individual metrics is defined in two metadata files packaged with the plug-in:

- The target type metadata file

  The content of the target type metadata file consists primarily of metric definitions, along with credential information and target properties. The fetchlet or receivelet that a metric will use is defined in the `QueryDescriptor` or `PushDescriptor` element within the target type metadata file.

  For information about key metric definition elements, see Overview of Key Metric Metadata Elements.

- The default collection metadata file

  The frequency at which data is collected for each metric is defined in the default collection metadata file. Metric Alert event conditions for each metric and the messages to display for such alerts are also defined in this file.

  For information about the default collection metadata file, see Creating the Default Collection File.

## Defining the Basic Response Metric Group

As a matter of practice, Oracle recommends that you specify at least a single `Response` metric group that includes the following metric for each target type:

- A `Status` metric that indicates target availability (required for all target types)

The corresponding default collection file must define a critical condition on the `Status` metric that represents the target status as up or down. For more information, see Defining Basic Metric Collection.

The following example (Response Metric) defines a `Status` metric. The return value of `Status` is as follows:

- 0: Target status is down
- 1: Target status is up

The process by which the metric data is collected is defined in the `QueryDescriptor` element. This descriptor specifies that the `OSLineToken` fetchlet invokes a Perl script (emrepresp.pl) to collect the data. The Perl script returns a standard out (stdout) data stream containing the collected data to the fetchlet.

Each property passed to the OSLineToken fetchlet execution is specified in a `Property` tag within the `QueryDescriptor` element.

- The OSLineToken fetchlet requires that a `GLOBAL` property called `command` be set to the command that is to be executed. Different tokens typically have specific required properties. For more information about the OSLineToken fetchlet, see OS Command Fetchlets.

- When a plug-in is deployed to a Management Agent, any scripts or binaries associated that were packaged within the /agent/scripts directory in the plug-in archive are written to the following directory in the Management Agent, where AGENT_HOME is the Management Agent plug-in home directory and *plugin_name* is the name of the plug-in:

  ```
  AGENT_HOME/plugins/plugin_name
  ```

  The `scriptsDir` property is a token that defines this location.

> **✎ Note:**
>
> In Enterprise Manager 11*g* and earlier, the scripts bundled with the plug-in were copied to the scripts directory under the Management Agent. However, for this release the scripts included in the plug-in are used directly.This changes the behavior of the scriptsDir property in the `QueryDescriptor` element. Previously, it referred to the directory under the Management Agent, but now it refers to the directory under the plug-in.If you want to refer to the scripts directory under the Management Agent, use the sdkScriptsDir property.

- The `script` property specifies the script (data_collector.pl) to be run.

  The EDK provides an example of this script:

  *edk*/samples/plugins/HostSample/demo_hostsample/stage/agent/scripts/emx/
  demo_hostsample

- The `startsWith` and `delimiter` properties specify the format of the STDOUT of the `script` executed. In this case, the script will return a single row that looks like this:

  em_result="*value for Load|value for Status*"

**Example: Response Metric**

```
<Metric NAME="Response" TYPE="TABLE">
 <TableDescriptor>
  <ColumnDescriptor NAME="Status" TYPE="NUMBER">
   <Display>
    <Label NLSID="oracle_emrep_resp_status">Status</Label>
   </Display>
  </ColumnDescriptor>
 </TableDescriptor>
 <QueryDescriptor FETCHLET_ID="OSLineToken">
  <Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>
  <Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property
  <Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl</Property>
  <Property NAME="script" SCOPE="GLOBAL">%scriptsDir%/emrepresp.pl</Property>
  <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
  <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
 </QueryDescriptor>
</Metric>
```

For a description of these elements, see Overview of Key Metric Metadata Elements.

## Defining Advanced Metrics

You can define much more complex metrics, such as metrics that collect CPU performance data or metrics for which values are computed from the values of other metrics. Examples of such advanced or complex metric definitions can be seen in the sample target type metadata file provided with the EDK:

*edk*/samples/plugins/HostSample/demo_hostsample/stage/oms/metadata/targetType/
demo_hostsample.xml

The following example shows a metric group containing metrics that collect CPU performance data. As in the previous example, the `QueryDescriptor` element specifies that the OSLineToken fetchlet will invoke the data_collector.pl script to collect the data.

**Example: Defining Advanced Metrics**

```
<Metric NAME="CPUProcessesPerf" TYPE="TABLE">
 <Display>
  <Label NLSID="hs_cpuprocessesperf_displayname">Host Process CPU Performance
   Statistics</Label>
  </Display>
  <TableDescriptor>
   <ColumnDescriptor NAME="ProcPID" TYPE="NUMBER" IS_KEY="TRUE">
    <Display>
     <Label NLSID="hs_cpuprocessesperf_procpid">PID</Label>
    </Display>
   </ColumnDescriptor>
  <ColumnDescriptor NAME="ProcUser" TYPE="STRING" IS_KEY="FALSE">
   <Display>
    <Label NLSID="hs_cpuprocessesperf_procuser">User</Label>
   </Display>
  </ColumnDescriptor>
  <ColumnDescriptor NAME="ProcCPU" TYPE="NUMBER" IS_KEY="FALSE">
   <Display>
    <Label NLSID="hs_cpuprocessesperf_proccpu">CPU Usage (%)</Label>
   </Display>
  </ColumnDescriptor>
  <ColumnDescriptor NAME="ProcCmd" TYPE="STRING" IS_KEY="FALSE">
   <Display>
    <Label NLSID="hs_cpuprocessesperf_proccmd">Command</Label>
   </Display>
  </ColumnDescriptor>
 </TableDescriptor>
 <QueryDescriptor FETCHLET_ID="OSLineToken">
  <Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property>
  <Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>
  <Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl</Property>
  <Property NAME="script" SCOPE="GLOBAL">%scriptsDir%/emx/demo_hostsample/data_
           collector.pl</Property>
  <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
  <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
 </QueryDescriptor>
</Metric>
```

The following example shows a test metric. The Management Agent can check some metrics to determine if a target has been specified correctly with valid instance properties. Setting the IS_TEST_METRIC attribute to "TRUE" provides a Test button when adding a target instance.

**Example: Defining a Test Metric**

```
<Metric NAME="Ping" TYPE="TABLE" IS_TEST_METRIC="TRUE" USAGE_TYPE="HIDDEN">
 <Display>
  <Label NLSID="label_metrics_ping">Ping Test</Label>
 </Display>
 <TableDescriptor>
  <ColumnDescriptor NAME="tcpIpPing" TYPE="NUMBER">
   <Display>
    <Label NLSID="test_ping">TCP Ping, Milliseconds</Label>
   </Display>
  </ColumnDescriptor>
```

```
</TableDescriptor>
<QueryDescriptor FETCHLET_ID="OSLineToken">
 <Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>
 <Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property>
 <Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl
 %sdkScriptsDir%/osresp.pl</Property>
 <Property NAME="ENVEM_TARGET_NAME" SCOPE="INSTANCE">hostName</Property>
 <Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
 <Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
 </QueryDescriptor>
</Metric>
```

The following examples show how to set `UnitCategory` and `Unit NLSID` on metric columns.

> **Note:**
>
> For a list of valid values for `UnitCategory` and `Unit NLSID`, see Metric Unit Standardization.

**Example 1: Setting `UnitCategory` and `Unit NLSID`**

```
<TableDescriptor>
<ColumnDescriptor NAME="cpuutil" TYPE="NUMBER" IS_KEY="FALSE"
TRANSIENT="FALSE" HELP="NO_HELP">
<Display>
<Label NLSID="olv_server_load_cpuutil">CPU Utilization (%)</Label>
<Description NLSID="olv_server_load_cpuutil_desc">CPU Utilization (%)</
Description>
<Unit NLSID="EM_SYS_STANDARD_UTILIZATION_PERCENTAGE">PERCENTAGE</Unit>
<UnitCategory>UTILIZATION</UnitCategory>
</Display>
```

**Example 2: Setting `UnitCategory` and `Unit NLSID`**

```
<ColumnDescriptor NAME="mem_avail" TYPE="NUMBER" IS_KEY="FALSE"
TRANSIENT="FALSE" HELP="NO_HELP">
<Display>
<Label NLSID="olv_server_load_mem_available">Memory Available (GB)</
Label>
<Description NLSID="olv_server_load_mem_available_desc">Memory
Available (GB)</Description>
<Unit NLSID="EM_SYS_STANDARD_DATASIZE_GB">GB</Unit>
<UnitCategory>DATA_SIZE</UnitCategory>
</Display>
```

**Example 3: Setting `UnitCategory` and `Unit NLSID`**

```
<ColumnDescriptor NAME="guest_vm_count" TYPE="NUMBER" IS_KEY="FALSE"
TRANSIENT="FALSE" HELP="NO_HELP">
<Display>
<Label NLSID="olv_server_guest_vms">Number of Guest VMs</Label>
<Description NLSID="olv_server_guest_vms_desc">Number of Guest VMs</
```

```
Description>
<Unit NLSID="EM_SYS_STANDARD_COUNT_NA">NA</Unit>
<UnitCategory>COUNT</UnitCategory>
</Display>
</ColumnDescriptor>
```

# Defining Repository Metrics

By default, the Management Agent collects metrics but you can define repository metrics. Repository metrics are collected at the Management Repository.

To define a repostitory metric, you must use the REPOSITORY attribute when defining the Metric element. For more information about the Metric element, see Table 3-4.

The following example provides an extract from a target metadata XML file where a repository metric is defined.

> **Note:**
>
> When defining repository metrics, you must set the metric TYPE attribute to TABLE. RAW is not supported for repository metrics.

**Example: Defining Repository Metrics**

```
<Metric NAME="Response"
        TYPE="TABLE"
        REPOSITORY="TRUE">
   <Display>
     <Label NLSID="REPOS_SQL">REPOS_SQL</Label>
   </Display>
   <TableDescriptor>
     <ColumnDescriptor NAME="Status" TYPE="NUMBER">
       <Display>
         <Label NLSID="Status">Status</Label>
       </Display>
     </ColumnDescriptor>
   </TableDescriptor>
   <QueryDescriptor FETCHLET_ID="REPOSITORY_SQL">
       <Property NAME="Type">SQL</Property>
       <Property NAME="Source">SELECT target_guid, 1 as Status from mgmt_targets
where target_type='tvmrtm200'</Property>
   </QueryDescriptor>
 </Metric>
```

# Categorizing Metrics

The purpose of the categorization of metrics is to define the nature of the data that is being collected. This is *only* for metrics that are visible in the **All Metrics** pages in the Cloud Control console.

Oracle recommends using the following guidelines when defining metric categories:

- Categorize all Metrics and Metric Groups visible to the user in the **All Metrics** page of the Cloud Control console.

- Use the published category values only as listed in Table 3-3.

- If all metrics in a metric group belong to the same category, then you can set the category at the metric group level and leave the category empty at the metric level. All metrics in the group will inherit this category. You can set one category only at the metric group level. For an example, see Defining a Category at Metric Group Level.

- If the metrics in the metric group have different categories, do *not* set a category at the metric group level. In such cases, set the category at the metric level. Each metric can have one category only. For an example, see the Defining a Category at Metric Level example.

The Metric framework has a *Default* metric class and a set of metric categories within this class. Table 3-3 provides a list of the available metric categories. You can categorize your metrics into the appropriate category within the Default metric class

**Table 3-3    Metric Categories**

| Category | Description |
|---|---|
| Availability | Lets you know whether a target or component is up. Used mainly for Response or Status metrics. |
| Capacity | Defines how much you have of something. |
| Fault | A severe error which results in a component not operating, or memory corruptions or data corruptions. |
| Load | How much work an entity is being asked to do. |
| LoadType | Indicates the characteristics of the work that an entity is being asked to do. |
| Response | How quickly is the system is responding. |
| BusinessKPI | Measures output in business terms. |
| Utilization | How much of something an entity is using |
| Security | Reporting on security issues. |

The following example categorizes all metrics within MyMetricGroup with a metric class of Default and a metric category of Load.

**Example: Defining a Category at Metric Group Level**

```
<Metric NAME="MyMetricGroup" TYPE="TABLE">
  <Display>     <Label>MyMetricGroup</Label>  </Display>
  <CategoryValue CLASS="Default" CATEGORY_NAME="Load"/>
  <TableDescriptor>
     <ColumnDescriptor NAME="MyMetric1" TYPE="NUMBER">    </ColumnDescriptor>
     <ColumnDescriptor NAME="MyMetric2" TYPE="NUMBER">    </ColumnDescriptor>
  </TableDescriptor> </Metric>
```

The following example categorizes the MyMetric1 metric with a metric class of Default and a metric category of Load, and the MyMetric2 metric with a metric class of Default and a metric category of Load Type.

**Example: Defining a Category at Metric Level**

```
<<Metric NAME="MyMetricGroup" TYPE="TABLE">
  <Display>
    <Label>MyMetricGroup</Label>
  </Display>  <TableDescriptor>
  <TableDescriptor>
```

```
      <ColumnDescriptor NAME="MyMetric1" TYPE="NUMBER">
        <Display>
          <Label>MyMetric1</Label>
        </Display>
        <CategoryValue CLASS="Default" CATEGORY_NAME="Load"/>
      </ColumnDescriptor>
      <ColumnDescriptor NAME="MyMetric2" TYPE="NUMBER">
        <Display>
          <Label>MyMetric2</Label>
        </Display>
        <CategoryValue CLASS="Default" CATEGORY_NAME="Load Type"/>
      </ColumnDescriptor>
    </TableDescriptor>
</Metric>
```

# Defining Adaptive Thresholds

Enterprise Manager provides the option to statistically compute metric thresholds that are adaptive in nature and this is available for all targets.

For more information about Adaptive Thresholds, see *Oracle Enterprise Manager Cloud Control Administrator's Guide*.

You can provide a named configuration for your target type. This quick configuration contains preconfigured metrics values and a computation mechanism.

To define a quick configuration:

1. Create a quick configuration file to determine the predefined values, and threshold calculation method for a group of metrics.

   The following example defines a quick configuration for the Host target type.

   **Example: Quick Configuration for Adaptive Thresholds**

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <baselineIntegration xmlns="http://www.oracle.com/EnterpriseGridControl/
   BaselineIntegration"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/
   BaselineIntg.xsd"
   targetType="host" version="12.1.0.4">
   <quickConfigration>
   <quickConfigDescription defaultText="Default text "
   resourceBundle="a.b.c.abcMsg" nlsId="SEC_CONFIG" />
   <baselineConfig configType="MV" subinterval="NX" interval="7"
   defaultText="Security Config" resourceBundle="a.b.c.abcMsg"
   nlsId="SEC_CONFIG">
   <metricSetting metricName="memfreePct" metricGroup="Load"
   thrsholdMethod="SIGLVL" warnignLvl=".99" criticalLvl=".999"
   numOccurrence="10" insufficientDataAction="UNSET" />
   <metricSetting metricName="cpuUtil" metricGroup="Load"
   thrsholdMethod="PCTMAX" warnignLvl="20" criticalLvl="30" numOccurrence="11"
   insufficientDataAction="UNSET" />
   metricSetting metricName="swapUtil" metricGroup="Load"
   thrsholdMethod="PCTMAX" warnignLvl="40" criticalLvl="50" numOccurrence="12"
   insufficientDataAction="UNSET" /
   </baselineConfig>
   <baselineConfig configType="MV" subinterval="NX" interval="21"
   defaultText="Space Config" resourceBundle="a.b.c.abcMsg"
   nlsId="SPACE_CONFIG">
   ```

```
<metricSetting metricName="memfreePct" metricGroup="Load"
thrsholdMethod="SIGLVL" numOccurrence="3" warnignLvl=".99"
criticalLvl=".999" insufficientDataAction="PRESERVE" />
<metricSetting metricName="cpuUtil" metricGroup="Load"
thrsholdMethod="PCTMAX" warnignLvl="25" numOccurrence="2"
criticalLvl="35" insufficientDataAction="PRESERVE" />
metricSetting metricName="activeMem" metricGroup="Load"
thrsholdMethod="PCTMAX" warnignLvl="45" criticalLvl="55" numOccurrence="5"
insufficientDataAction="PRESERVE" /
<metricSetting metricName="memUsedPct" metricGroup="Load"
thrsholdMethod="PCTMAX" warnignLvl="65" criticalLvl="75" numOccurrence="2"
insufficientDataAction="PRESERVE" />
</baselineConfig>
</quickConfigration>
</baselineIntegration>
```

2. Register the quick configuration file with Enterprise Manager by placing the file in the following location before plug-in deployment:

   *plugin_stage*/oms/metadata/adaptiveThreshold/

   For information about plug-in deployment, see Validating, Packaging, and Deploying the Plug-in .

3. The end user activates the quick configuration from the Cloud Control UI (From the **Metric and Collection Settings** page, select **Advanced Threshold Management**).

   For information about using the Advanced Threshold Management feature, see the *Oracle Enterprise Manager Cloud Control Administrator's Guide*.

# Overview of Key Metric Metadata Elements

**Table 3-4    Key Elements Used to Define Metrics**

| Element | Description |
|---|---|
| Metric | Required. Defines a metric group containing one or more metrics, each defined in a `ColumnDescriptor`. The `Metric` element includes the following attributes:<br><br>• `NAME`: The name of the metric group, up to 64 characters long.<br><br>• `TYPE`: Valid values are "TABLE" or "RAW". Typically set to "TABLE". Do not modify the type of the metric after creation.<br>TABLE - Metric data is loaded into generic tables. Usually these metrics are performance or usage metrics. They can use the Enterprise Manager core infrastructure for charting and display on the Cloud Control console.<br>RAW - Metric data is loaded into the table specified in the metric definition. The name of this table must be the name of a valid table in the SYSMAN schema. Updatable views are not allowed. Do not modify STORAGE_TABLE_NAME or STORAGE_COLUMN_NAME.<br><br>• `REPOSITORY`: Specifies whether the metric is collected by Management Agent or at the Management Repository. Valid values are:<br>TRUE: Metrics are collected at the Management Repository.<br>FALSE: Metrics are collected by the Management Agent (default). If you do not include this attribute, then the defined metric is collected by the Management Agent.<br><br>• `IS_TEST_METRIC`: The Management Agent can check some metrics to determine if a target has been specified correctly with valid instance properties. This attribute marks this metric as one of the test metrics. By default, the value is set to "FALSE". Setting this value to "TRUE" provides a Test button when adding a target instance.<br><br>• `USAGE_TYPE`: Specifies if the metric is viewable.<br>`VIEW_COLLECT`: If `USAGE_TYPE=VIEW_COLLECT`, then the metric will appear on the Enterprise Manager Cloud Control UI.<br>`COLLECT_UPLOAD`: If `USAGE_TYPE=COLLECT_UPLOAD`, then the metric will be displayed on the **Metric Collection Settings** page for modifying the collection schedule.<br>`HIDDEN`: If `USAGE_TYPE=HIDDEN`, then the metric is not viewable and the Management Agent will not upload data collected by the metric to the Management Repository.<br>**Note**: Do not modify the `USAGE_TYPE` of a metric. |
| TableDescriptor | Required when the Metric `TYPE` attribute is set to "TABLE". It contains one or more ColumnDescriptor elements, each defining a metric to collect |

**Table 3-4 (Cont.) Key Elements Used to Define Metrics**

| Element | Description |
|---------|-------------|
| ColumnDescriptor | Defines a single metric to be collected. It includes the following attributes: |

- `NAME`: The name of the metric, up to 64 characters long.
- `TYPE`: Describes how the metric data will be stored in the Management Repository. The value is either "NUMBER" or "STRING".

  **Note**: Nested tables are not supported
- `IS_KEY`: Indicates if the metric is to be treated as a primary key column in the Management Repository. Values are "TRUE" or "FALSE" (default).

  **Note**: Do not change the number or the order of key columns across meta versions. Oracle recommends using a stable key value. Do *not* use a line number, or time stamp, or session ID as a key value
- `TRANSIENT`: The metric will not be uploaded to the Management Repository. Use this attribute for calculating rate data. Values are "TRUE" or "FALSE" (default). For more information, see Defining Target Metadata.
- `COMPUTE_EXPR`: This attribute specifies a formula for calculating the value of the column. Columns previously defined in the Table descriptor can participate in the calculation. Attaching an underscore (_) prefix to a column name denotes the previous value of a column.

  For a list of predefined special values, see ColumnDescriptor of Enterprise Manager DTD .

**Table 3-4    (Cont.) Key Elements Used to Define Metrics**

| Element | Description |
|---|---|
| `QueryDescriptor` | Defines a command to run, which returns the set of data described in the `TableDescriptor`. The element contains one or more `Property` elements, each defining a property to pass in with the command invocation.<br><br>**Note**: You can refer to earlier defined properties using the `%property_name%` format. For example:<br><br>`<Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>`<br>`<Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl</Property>`<br><br>The element includes a `FETCHLET_ID` attribute that identifies the fetchlet mechanism that will be used to process the request. Properties required for the fetchlet invocation are specified in one or more `Property` elements within the `QueryDescriptor`.<br><br>The following are among the fetchlets commonly used by plug-in developers:<br><br>• `OS`: Executes the given operating system command or script and returns the command's output in a single cell table.<br>• `OSLines`: Similar to the fetchlet, but returns the output tokenized by lines.<br>• `OSLineToken`: Similar to the fetchlets, but the output is tokenized first by lines; each line is then tokenized by a given delimiter set.<br>• `HTTPDataLineToken`: Invokes an HTTP request to the specified URL.<br>• `SQL`: Executes the specified SQL script against the specified Oracle Database.<br>• `Snmp`: Invokes the specified SNMP call to the specified SNMP agent.<br>• `WBEM`: Invokes the specified WBEM call to the specified CIMOM object repository.<br><br>For a complete list of available fetchlets and information about their usage, see Using Fetchlets .<br><br>The SCOPE property defines where the property value is to be obtained. The following scope options are available:<br><br>• SCOPE="GLOBAL". Obtain the property value from other variables defined within the current target type metadata file. This includes constants, such as the "\|" shown in the Sample plugin_registry.xml File example.<br>• SCOPE="INSTANCE". Obtain the property value from instance properties.<br>• SCOPE="ENV*xxx*". Obtain the property value from an environment variable "*xxx*".<br>• SCOPE="SYSTEMGLOBAL". Obtain the property value from the emd.properties file located in the $AGENT_HOME/sysman/config directory.<br>• SCOPE="USER". Obtain the property value from the collector or user. |

**Table 3-4    (Cont.) Key Elements Used to Define Metrics**

| Element | Description |
| --- | --- |
| PushDescriptor | Defines object identifiers (OIDs). The Management Agent uses an SNMP receivelet to listen for SNMP traps as defined in the target type's Management Information Base (MIB). The `PushDescriptor` is part of the metric definition of a receivelet. |
| | Defines a command to run, which returns the set of data described in the `TableDescriptor`. The element contains one or more `Property` elements, each defining a property to pass in with the command invocation. |
| | **Note**: You can refer to earlier defined properties using the `%property_name%` format. For example: |
| | ``` <Property NAME="MatchSpecificTrap" SCOPE="GLOBAL">50</Property> <Property NAME="MatchAgentAddr" SCOPE="INSTANCE">AdminAddress</Property> ``` |
| | The element includes a `RECVLET_ID` attribute that identifies the receivelet mechanism that will be used to process the request. Properties required for the receivelet invocation are specified in one or more `Property` elements within the `PushDescriptor`. |
| | The SNMP receivelet is the only receivelet commonly used by plug-in developers. For more information about the SNMP receivelet, see Using Receivelets . |
| | The SCOPE property defines where the property value is to be obtained. The following scope options are available: |
| | • SCOPE="GLOBAL". Obtain the property value from other variables defined within the current target type metadata file. For example, if a Match* property is GLOBAL-scoped, then it determines with which metric the trap is associated. |
| | • SCOPE="INSTANCE". Obtain the property value from instance properties. For example, if the MatchAgentAddr property is an INSTANCE-scoped property, then it determines to which target instance the trap belongs. |
| | For examples of SCOPE property definitions, see the examples in Using Receivelets . |
| ElementDescriptor | Used to compute aggregation metrics. Specifies the execution plan for evaluating a metric. The Management Agent runs each statement of the plan, in the order it is defined, to produce a Metric Result. The Metric Result generated as result of the evaluation of the last statement of the execution plan will be returned. |

Table 3-4 describes the key elements that define metrics. For additional information about defining metrics, see Guidelines for Creating Target Metadata.

## Troubleshooting Metric Definitions

This section provides some troubleshooting tips if you encounter any issues with your metric definitions.

- To list metrics for a given target type, from a SQL*Plus session, enter the following:

  ```
  select * from mgmt_metrics where target_type=target_type and
  type_meta_ver=max_type_meta
  ```

- To check if there are any metric load issues, from a SQL*Plus session, enter the following:

  ```
  select * from sysman.mgmt_system_error_log where module_name='METRIC_LOAD'
  ```

- To check if there are any collection issues during the metadata load, from a SQL*Plus session, enter the following:

```
select * from sysman.mgmt_system_error_log where
module_name='MGMT_COLLECTION.Collection Subsystem'
```

* To check what is being collect for a specific target, from a SQL*Plus session, enter the following:

```
select * from sysman.gc_metric_values where entity_type=target_type and
entity_name=target_name
```

# Creating the Default Collection File

The default collection metadata file for a target type defines the following:

* The metric data (including configuration collection metric data) to be collected from targets and written to the Management Repository
* The frequency of at which this metric data is collected
* Thresholds that, when exceeded, will cause a Metric Alert event to be raised
* An optional message to display when a threshold is exceeded

> **✏ Note:**
>
> For a default collection file, Oracle mandates that you use the same file name as the target type metadata file name.
>
> For example, the target type metadata file present under `oms/metadata/targetType/demo_hostsample.xml` in HostSample plug-in must have the corresponding default collection file with the same name under `oms/metadata/default_collection/demo_hostsample.xml`.
>
> For information about naming the target type metadata file, see Naming the Target Type Metadata File.

Note that the value of the `TYPE` attribute and the `META_VER` attributein the default collection metadata file *must* match the `TYPE` and the `META_VER` values defined in the target type metadata file to create an association between them.

As noted, you can also specify Metric Alert event conditions on each metric that will be raised as Incidents within Enterprise Manager Cloud Control. Such events are generated when a threshold specified in this file is exceeded. For example, you may want to raise a WARNING alert when CPU usage reaches 90% of capacity. You can also specify the message text to be displayed in Enterprise Manager Cloud Control when an alert event is triggered.

The EDK includes an example of a default collection file in the following location:

```
/samples/plugins/HostSample/stage/oms/metadata/default_collection/
demo_hostsample.xml
```

For information about defining the elements in the default collection file, see Overview of Key Default Collection Metadata Elements and Guidelines for Creating Target Metadata.

# Grouping Similar Metrics For Collection

For efficiency, metrics are typically grouped together for collection, enabling certain metrics to be collected at the same time or same frequency. This is useful because it guarantees the order of execution of the metrics, which is important if some metrics rely on the results of other metrics.

Each group of metrics to be collected together is defined in a CollectionItem within the default collection file. The collection schedule for the group is defined in a Schedule element. The CollectionItem provides the end user with the most control to enable or disable metrics and to change the collection schedule.

Each metric included in the group is in turn defined within a MetricColl element within the CollectionItem. (Note that if the CollectionItem contains just a single metric, like the Response metric example shown in Defining the Basic Response Metric Group, it is not necessary to specify the MetricColl tag.)

Note that the UPLOAD value for the CollectionItem is set to 6, meaning that every sixth collection of data will be written to the Management Repository. Because the IntervalSchedule specifies that data will be collected every 5 minutes, the data will be written to the Management Repository every 30 minutes (or every sixth data collection).

```
<TargetCollection>
...
 <CollectionItem NAME="Perf" UPLOAD="6">
  <Schedule>
   <IntervalSchedule INTERVAL="5" TIME_UNIT="Min"/>
  </Schedule>
  <MetricColl NAME="CPUProcessesPerf">
   ...
  </MetricColl>
  <MetricColl Name="MemoryPerf">
   ...
  </MetricColl>
 </CollectionItem>
...
</TargetCollection>
```

You should consider grouping metrics into a CollectionItem if any of the following applies:

- The metrics are logically related, such as all metrics related to performance

- The metrics should be collected at the same frequency, such as all metrics that should be collected every 5 minutes

- The metrics should be collected at roughly the same time, such as metrics collected during non-peak times

- You want to collect all of the metrics, or none of the metrics, at the same time

Note that if you have metrics that will be collected on demand, grouping them will improve performance and reduce the communications required by the Management Agent and Oracle Management Service to collect and return metric data from the target.

Providing different collection intervals will provide the best flexibility to users to schedule each metric independently. Grouping together unrelated metrics is not

advisable, as you will not have the ability to turn off collection of just a few metrics in the group without disabling those metrics that you do need.

# Defining Basic Metric Collection

The following represents the `CollectionItem` entry for the basic `Response` metric group, which includes the `Status` metric. It specifies that data for this metric should be collected every 5 minutes, which is the standard collection interval for this type of metric.

A condition has been set on the `Status` metric. For more information about alert conditions, see Creating the Default Collection File and Table 3-5.

Note that because the `CollectionItem` contains just one metric (`Status`), it is not necessary to include a `MetricColl` tag for the single metric.

```
<TargetCollection META_VER="2.0" TYPE="test_demo_targetType">
 ...
 <CollectionItem NAME="Response">
  <Schedule>
   <IntervalSchedule INTERVAL="5" TIME_UNIT="Min"/>
  </Schedule>
  <Condition COLUMN_NAME="Status" CRITICAL="0" OPERATOR="EQ"
CLEAR_MESSAGE_NLSID="Response_Status_clearalertmessage"
  MESSAGE="Failed to connect to database instance:
%oraerr%."MESSAGE_NLSID="Response_Status_alertmessage"/>
 </CollectionItem>
...
</TargetCollection>
```

# Defining Advanced Metric Collection

The following example illustrates the collection of a more advanced metric that raises a metric alert when specified `WARNING` and `CRITICAL` thresholds are exceeded. These thresholds, and the message to send to Cloud Control when they are exceeded, are defined in the `Condition` element.

The data for each metric is specified in a `MetricColl` element within a `CollectionItem`, as shown in this example. For a description of the elements in this example, see Table 3-5.

**Example: Defining Advanced Metric Collection**

```
<TargetCollection>
...
 <CollectionItem NAME="Perf" UPLOAD="6">
  <Schedule>
   <IntervalSchedule INTERVAL="5" TIME_UNIT="Min"/>
  </Schedule>
  <MetricColl NAME="CPUProcessesPerf">
   <Condition COLUMN_NAME="ProcCPU" WARNING="75" CRITICAL="90" OPERATOR="GE"
              OCCURRENCES="2"
              MESSAGE="The value for %columnName% is %value%%, which is above
the
              critical (%critical_threshold%%) or warning (%warning_threshold%%
%)
              threshold."
              CLEAR_MESSAGE="The value for %columnName% is %value%%, which is
              below the critical (%critical_threshold%%) or warning (%warning_
```

```
                threshold%%%) threshold." />
  </MetricColl>
 </CollectionItem>
...
</TargetCollection>
```

Note that in addition to a message sent to Enterprise Manager when either the
WARNING or CRITICAL thresholds are passed, and "all clear" message to be sent
when an alert condition no longer exists has also been defined in the CLEAR_MESSAGE
attribute.

## Defining Target Configuration Data Collections

As with all other metrics, the frequency at which the configuration metric data is
collected is defined default collection file. Given the size of target configuration
collections and the infrequent change rate, these metrics should ideally be collected
every 24 hours, during off-peak hours.

Note that the value of the TARGET_TYPE attribute of the root METADATA SNAP_TYPE
attribute in the configuration metadata file must be identical the TYPE attribute of
TargetCollection in the default collection file.

The following example defines the collection frequency for the HostConfig metric

```
<TargetCollection>
...
<CollectionItem NAME="HostSampleSnap" CONFIG="TRUE">
    <Schedule OFFSET_TYPE="INCREMENTAL">
      <IntervalSchedule INTERVAL="24" TIME_UNIT="Hr"/>
    </Schedule>
    <MetricColl NAME="HostConfig" />
     </CollectionItem>
...
</TargetCollection>
```

## Overview of Key Default Collection Metadata Elements

Table 3-5 describes the key elements included in the default collection metadata file.

**Table 3-5    Key Elements Within the Default Collection Metadata File**

| Element | Description |
| --- | --- |
| TargetCollection | Required. The root element for the file. It includes a TYPE attribute and the META_VER attribute that must match the TYPE attribute and the META_VER attribute of the TargetMetadata element in the target type metadata file. |

**Table 3-5    (Cont.) Key Elements Within the Default Collection Metadata File**

| Element | Description |
|---|---|
| CollectionItem | Defines a collection frequency and threshold values for a set of metrics. The frequency defined in the included Schedule element will apply to all metrics in the collection group. The element includes the following attributes:<br><br>• NAME: Defines the set of metrics to be collected as a group.<br>• UPLOAD: Specifies what *n*th data collection is written to the Management Repository. The default is 1, which means that performance data is uploaded every time it is collected. In Response Metric example, every 6th data collection is uploaded to the Management Repository.<br>• UPLOAD_ON_FETCH: When set to TRUE, the first configuration collection is uploaded to the Management Repository immediately. All subsequent collections are bundled. By default, this value is set to FALSE and the UPLOAD_ON_FETCH attribute is ignored. |
| Schedule | Contains an IntervalSchedule element defining the collection frequency for a CollectionItem. It includes the following attributes:<br><br>• INTERVAL: The collection frequency.<br>• TIME_UNIT: The unit of time (such as Min for minutes) that the value of INTERVAL corresponds to.<br><br>**Note**: Response metrics must be collected frequently. Define the collection interval at a value between 1 minute and 5 minutes. Other metrics should be collected less frequently. Usually an interval of 15 minutes is sufficient. However, if you are defining metrics collecting a lot of data, then consider an interval of 30 minutes or 1 hour. |
| MetricColl | Contains one or more Condition elements corresponding to a single metric group defined in a Metric element in the target type metadata file. Each condition must have an associated alert message that includes a metric description with a specified unit and the %value% placeholder for the value that causes the threshold to be raised.<br><br>The NAME attribute in this element must match the NAME attribute in the corresponding Metric element. |

**Table 3-5    (Cont.) Key Elements Within the Default Collection Metadata File**

| Element | Description |
| --- | --- |
| Condition | Defines a metric alert condition. It contains the following optional attributes: |
| | • COLUMN_NAME: The name of a metric defined in a ColumnDescriptor element in the target type metadata file. The value of this attribute must match the NAME attribute of the ColumnDescriptor element. |
| | • WARNING: Defines the threshold at which a "warning" condition exists. A metric alert will be generated when this value is exceeded, which will include the text specified in the MESSAGE attribute. |
| | In most cases, to allow users to set the threshold value, set this attribute to "NotDefined". |
| | • CRITICAL: Defines the threshold at which a "critical" condition exists. A metric alert will be generated when this value is exceeded, which will include the text specified in the MESSAGE attribute. |
| | In most cases, to allow users to set the threshold value, set this attribute to "NotDefined". |
| | • OPERATOR: Determines how to apply the threshold values specified in the CRITICAL and WARNING attributes. In the Defining Advanced Metric Collection example, `GE` specifies that the Warning threshold occurs when ProcCPU is greater than or equal to 75 and the Critical threshold occurs when ProcCPU is greater than or equal to 90. Other values include: |
| | LE: Less than or equals |
| | EQ: Equals |
| | LT: Less than |
| | GT: Greater than |
| | NE: Not equal |
| | CONTAINS: True if the second argument is a substring of the first string. |
| | MATCH: True if the first argument (regular expression) matches the second argument. |
| | • OCCURENCES: Defines the number of successive metric collections that must be returned with the Warning or Critical threshold exceeded before the warning or critical condition is triggered. In most cases, set this value to 2 to avoid alerting on spikes. |
| | • MESSAGE: Contains a message to display when the thresholds specified in the WARNING or CRITICAL attributes have been exceeded. |
| | The built-in message attributes such as %columnName% and %critical_threshold% are embedded in the string; the appropriate value will be substituted when the message is generated at runtime. For example: "The value for %columnName% is %value%%%. It has fallen below the critical (%critical_threshold%%%) or warning (%warning_threshold%%%) threshold." |
| | • CLEAR_MESSAGE: Contains an "all clear" message that will be displayed when the value of the metric returns to a "non-alert" value; that is, when it drops below the thresholds indicated in WARNING and CRITICAL. |

## Troubleshooting the Collection Process

This section provides some troubleshooting tips if you encounter any issues with your collection process.

To check if the collection is disabled for a specific target, from a SQL*Plus session, enter the following:

```
select * from sysman.mgmt_collections where object_type=2 and
object_guid=target_guid and is_enabled=1
```

# Guidelines for Creating Target Metadata

When developing target type definition files for new plug-ins, special consideration must be paid to the way in which you want a particular target type to be monitored. How a target type is monitored can greatly affect Enterprise Manager performance. Follow these general guidelines for defining target metadata and collections to optimize system performance.

## Defining Target Metadata

Metadata is data about data. Generically, the term refers to any data used to help the identification, description, and location of a network entity. Target metadata for an Enterprise Manager target consists of the metrics a user wants to expose and the methods used to compute those metrics.

> **Note:**
>
> Ensure that all metrics that are viewable are categorized. At a minimum, metrics that have thresholds must be categorized so that generated incidents are categorized. For more information about categorization, see Categorizing Metrics.

- **Metadata Version**

  Whenever the target metadata changes, increment the metadata version (META_VER). For more information, see Defining the Target Type Metadata.

- **Real-time Only Metrics**

  Performance metrics can be classified into metrics that must be computed to track performance trends and others that are more useful to drill down to get the details at a particular point in time. Real-time only metrics include those that need contextual information to return detailed information about a particular subset of the system, such as the resource utilization for specific processes, to diagnose further.

- **Choice of Key Columns**

  A key column in a metric is used in the management repository to trend performance data on an axis, such as the tablespace usage per database tablespace. Key-based metrics should be used to model sub-components of the target for which meaningful metric data should be collected, either for target

monitoring or target diagnostic purposes. As such, only key columns that are the logical identifiers of the target sub-components should be included in the metric.

Note that including key columns for which the number of distinct values collected across a large number of targets could result in an excessive number of key values being stored in the management repository. For example, using a timestamp (or equivalent, like database SCN or UNIX ctime) as a key value will result in a new value for every collection for every target, and is therefore not advisable.

Including a combination of key columns can also be problematic. For example, if you include three key columns in a metric, in which each key can take one of 10 target-specific values (10X10x10) multiplied by the number of targets, you would be collecting data for 1000 keys per target. This could be considered excessive if more than a handful of targets are being managed.

Do not define metrics that have nonsharable and durable keys across targets and time.

You do not have to have key columns, but the query descriptor must return a single row.

> **Note:**
>
> Do *not* modify key columns, that is, order, data type, or number after creating a metric.

- **Transient Columns**

  In some cases, metric columns can be used to compute the values of other more interesting metric columns. When the original columns are not required, then you can mark these columns as transient so that they are not uploaded to the Management Repository, therefore saving space.

  Metrics that are dependent on a duration of time must not be uploaded. Mark these types of metrics as transient. This includes delta metrics such as Request Process (last collection interval).

- **Rate Metrics**

  Metrics should contain data values for recent activity. In many cases, to do this, you must create rate metrics out of existing metrics. The COMPUTE_EXPR attribute (defined in Table 3-4) specifies the formula for calculating the value of a column. The following list provides the supported grammar for Compute Expression:

```
expression := (cond_expr | (cond_expr ? cond_expr : cond_expr)
cond_expr := (string_expr |
(string_expr == string_expr) |
(string_expr < string_expr) |
(string_expr > string_expr) |
(string_expr <= string_expr) |
(string_expr >= string_expr) |
(string_expr __contains string_expr) |
(string_expr __beginswith string_expr) |
(string_expr __endswith string_expr) |
(string_expr __matches string_expr) |
(string_expr __delta string_expr))
string_expr := (simple_expr |
```

```
(simple_expr __leadingchars simple_expr) |
(simple_expr __trailingchars simple_expr) |
(simple_expr __substringpos simple_expr))
simple_expr := (term |
(simple_expr + term ) |
(simple_expr - term) )
term := (unary_expr |
(term * unary_expr ) |
(term / unary_expr ) )
unary_expr := (factor |
(__is_null factor) |
(__length factor) |
(__to_upper factor) |
(__to_lower factor) |
(__ceil factor) |
(__floor factor) |
(__round factor) )
factor := ( identifier |
string_literal |
number |
'(' expression ')' )
string_literal := '\'' (character | "\\'" )* '\''
```

To create rate metrics from existing metrics, define the following metrics:

– Calculating Delta

```
requests.completed.delta = (requests.completed > _requests.completed) ?
(requests.completed - _requests.completed) : 0
```

If the current value of requests.completed is more than the previous value, then obtain delta by getting the difference between the current value and the last value. Otherwise return 0.

– Calculating Rate

```
requests.completed (per minute) = (requests.completed >
_requests.completed) ? (requests.completed - _requests.completed) *60 /
__interval: 0
```

If the current value of requests.completed is more than the previous value, then obtain delta by getting the difference between the current value and the last value and then multiplying by 60 and dividing by the interval between 2 collections. Otherwise, return 0.

- **Metrics and Microsoft Windows**

  When creating metrics for custom targets, it is important to take into account the cost (CPU usage) of creating additional operating system (OS) processes. This is especially true for systems running Microsoft Windows where process creation is much more CPU intensive compared to UNIX-based systems such as Linux or Oracle Solaris. The percentage CPU utilization increases linearly with creation of child processes. To minimize process creation, avoid executing OS programs or commands from metric collection scripts. For example, when writing Perl scripts, avoid using the system function or backticks (``) to execute an OS command.

- **Target Properties (Static Versus Dynamic)**

  Target properties are named values that can be used for computing the metrics of the target, or for display in the home page of the target. The list of target properties is specified in the metadata to allow data driven user interfaces to register targets, and for the Management Agent to validate that a target instance is complete.

- **Static Instance Properties**: These are properties whose values need to be specified for a target in the targets.xml entry for the target. An instance property can be marked optional if the target declaration is considered complete even without the specification of the property. The metadata specification of a target property can also provide a default value for use in a configuration user interface.

- **Dynamic Instance Properties**: The Management Agent also allows for target instance properties to be computed. Such properties are computed using a QueryDescriptor very similar to the ones used in metrics.

  Use of dynamic properties reduces the work involved in configuring a target by allowing certain properties to be computed rather than requiring the user to correctly specify their values (for example, the Version property of a database can be reliably computed given addressing information).

  The Management Agent allows for the fact that the target needs to be up for the successful computation of these dynamic properties by recomputing the properties each time a target bounce is detected (each time the target status changes to **Up**).

- **Metrics**

  The metric concept, as it pertains to the Management Agent, can be used to denote configuration and performance information.

  - **Configuration Metrics**: Configuration metrics collect data similar to target properties that denote the configuration of the target. This information is periodically refreshed and can be used to track changes in the setup of a target. The collection interval on such metrics is typically on the order of about 24 hours.

  - **Performance Metrics**: Performance metrics are used to track the responsiveness of a target. These metrics are typically collected more often than configuration metrics though the interval of some performance metrics may vary widely from those of others. Also, performance metrics usually ship with thresholds that are the basis of performance alerts for the target.

    A required metric for all targets is the "Response" metric consisting of a "Status" column with a condition on it. This metric is used to track the availability of the target.

  The conventions used in naming your metrics are extremely important because many areas of the Enterprise Manager user-interface are data-driven. For example, actual metric column labels and key values can be part of the page title, instruction text, or column headings. Specifically, these elements appear on the **All Metrics** page, **Metric and Collections Settings** page, **Event Rules** page, **Group Charts** page, and other pages within the Enterprise Manager user interface. For this reason, Oracle recommends the following metric naming conventions.

  - Ensure that metric column names are as explicit as possible. Do not include *count* in the column name because it adds to the length of the name and does not provide value to the end user. For example:

    Errors (per minute)

    **Note**: Do not use Error Count (per minute).

  - All metric column names (labels) must be unique within a given target type and version, and easily understood by the user (metric units used as required). If the metric refers to a unit of measure, include the unit in the metric label inside a parenthesis. For example:

**ORACLE**

Network Interface Total I/O Rate (MB/sec)

Requests Processed (per minute)

Transactions Committed (%)

– Metrics that are increasing values which reset at startup should not be uploaded. Mark these metrics as transient and include since startup in the metric label inside a parenthesis. For example:

Processing Time (since startup)

Errors (since startup)

Requests (since startup)

Average Execution Time (ms - since startup)

– All metric column names (labels) should be self-explanatory without dependence on the metric name. For example:

Tablespace Space Used (%)

– Key column names must be self-explanatory. Enterprise Manager uses these names when specifying metric thresholds or configuring notifications. For readability, the name of the key column name must fit easily within the phrase "all *key column name* objects". For example:

all tablespace objects

– Across target versions, the same columns must use the same labels. This ensures columns, such as metric columns and short names, have the same NLS IDs across different target versions.

# Defining Collections

Collections are the mechanism by which the Management Agent periodically computes the metrics of a target and uploads the data to the Management Repository. The most important thing to keep in mind when creating the collections for a target type is to avoid overburdening the Management Repository with excess data. In a large enterprise with hundreds of Management Agents and thousands of targets, the key to scalability is to limit the amount of data collected about a target that is uploaded to the repository. This is especially important since raw data is maintained for 24 hours - rollup benefits only accrue beyond that point.

# Alert Message Guidelines

Alert messages tell the user when something is wrong. These messages should also assist the user in solving the problem. Oracle recommends following these content guidelines when writing alert messages:

• Alert messages should be meaningful, and must include the metric display name, metric value and the thresholds that caused the alerts.

The most significant part of the message should be covered within the first 60 characters of the message text. The reason is that the first part of the message is the most visible to users in e-mail notifications, incident tables containing the alert message, and so on.

• Include warning thresholds and critical thresholds in the alert message.

- Target down messages should, in addition to indicating that the target is down, include information indicating possible reasons why the target may be down.

- Include error codes and messages whenever possible.

The following is a good example of an alert message:

```
CPU Load (Run Queue Length averaged over 15 minutes) is %value%, crossed warning
(%warning_threshold%) or critical (%critical_threshold%) threshold.
```

Note that you should not include information on how to resolve or diagnose the problem in the alert message. You should instead provide this content in the Guided Resolution section of Incident Manager. See Providing Content in the Guided Resolution Region for more information.

## Metric Evaluation Order

It is important to pay attention to metric evaluation order so as to avoid metric collection failures. For example, the Response metric should be evaluated first in order to prevent a collection failure when a target is down. Also, ensure that the metric collection error is consistent. For example, you should have a new message every time that a metric is collected.

When a `CollectionItem` tag is used to define a collection, then the Management Agent evaluates all metrics with a collection item in order. However, collection items run independent of each other.

> **Note:**
>
> Programmatic logic of the Management Agent distributes the metric evaluations so that each evaluation is separated by approximately 10 seconds.

## Collection Frequency

In general, there is almost never a good reason to collect information at intervals smaller than 5 minutes. In the rare case where data variations occur at a smaller granularity and administrators need to be notified sooner, the Management Agent provides the capability to use a small collection interval to compute the metrics and threshold information while still only uploading data once in every *nn* computation cycles.

## Controlling Number of Rows

Some metrics can result in the creation of a large number of rows in a Management Repository table. In some cases, only a subset of these rows may need to be uploaded to the repository. The Management Agent allows the specification of filter conditions that can be used to find rows to skip uploading. Also, a "limit_to" clause can be used on metrics that return sorted metric data to upload only the first n rows to the repository.

# Localizing Target Metadata

To localize your target metadata:

- Read About Target Metadata Localization
- Define the Resource Bundle Package
- Localize Metric Messages
- Package Resource Bundles

## About Target Metadata Localization

Target metadata can optionally support localized strings, including target type display name and metric and metric column labels, enabling Enterprise Manager to display labels in the language and locale of each Enterprise Manager user. To support this feature, the target metadata file must include the RESOURCE_BUNDLE_PACKAGE property in the TargetMetadata tag. The RESOURCE_BUNDLE_PACKAGE property specifies the location of the resource properties files that contains the localized target strings. For information about the TargetMetadata tag, see Creating the Target Type Metadata File.

## Define the Resource Bundle Package

Use the three-part plug-in id, followed by the package selected for the resource bundles. For example, if the plug-in ID is test.group.domain, then define the resource bundle package as follows:

```
RESOURCE_BUNDLE_PACKAGE=test.group.domain.rsc
```

In the previous example, `rsc` is the package selected for the resource bundles. You can use any alphanumeric string for the package name but you cannot include special characters.

The strings included in the target metadata that can be externalized to the resource properties file are the Label tags associated with the target type, metric and column items.

> **✎ Note:**
>
> If the resource property cannot be loaded, then the Label tag has a default value that is displayed and the NLSID property specifies the key to be used to load the string resource that will be loaded in the user's locale.

You must place all of the strings defined for your target metadata in a resource properties file, named `target_typeMsg.properties`. Include this file in the corresponding directory in the resource deployment. For more information, see Package Resource Bundles.

In the following example, the plug-in ID is test.group.domain and the target type is domain_widget.

**Example: Defining a Resource Bundle Package for Target Metadata Localization**

```
<TargetMetadata META_VER="1.0" TYPE="domain_widget"
RESOURCE_BUNDLE_PACKGE="test.group.domain.rsc">
<Display>
<Label NLSID="dom_widget">Domain Widget</Label>
</Display>
```

For this plug-in deployment, you must have a resource properties file named
test.group.domain.rsc.domain_widgetMsg.properties. This file contains all the strings
for the target metadata and includes the following:

```
# Strings for the domain_widget target type within the test.group.domain plug-in
dom_widget = Domain Widget
```

## Localize Metric Messages

In the default collection metadata file, metric collection conditions can specify the
following properties for the message alert and cleared message:

| NLSID | Description |
|---|---|
| MESSAGE | MESSAGE_NLS_ID |
| Specifies the default message (in English) when a condition is met. | Specifies the resource identifier that will be used to locate the translated version of the message in the resource properties file associated with the target type metadata. |
| CLEAR_MESSAGE | CLEAR_MESSAGE_NLS_ID |
| Specifies a cleared message to be sent when an alert condition no longer exists. | Specifies the resource identifier that will be used to locate the translated version of the cleared message in the resource properties file associated with the target type metadata. |

The following example provides an example of a metric definition that includes the
resource identifier for the alert message and cleared message.

**Example: Defining a Metric to Include Localization Properties**

```
<MetricColl NAME="CPUPerf">
  <Condition COLUMN_NAME="non_nice" WARNING="NotDefined" CRITICAL="NotDefined"
     OPERATOR="GE"
     MESSAGE="The value for %columnName% is %value%%.  It has risen above the
critical (%critical_threshold%%) or warning (%warning_threshold%%) threshold."
     MESSAGE_NLS_ID="dhs_non_nice_cond_msg"
     CLEAR_MESSAGE="The value for %columnName% is %value%%.  It has fallen
below the critical (%critical_threshold%%) or warning (%warning_threshold%%)
threshold."
     CLEAR_MESSAGE_NLS_ID="dhs_non_nice_clear_msg" />
```

## Package Resource Bundles

Before you package the resource bundle, check the About Resource Property Bundle
Content to ensure the contents of your package are formatted correctly.

## About Resource Property Bundle Content

All of these resource properties files must be formatted as proper Java resource properties bundles. Include the appropriate country and locale according to the following Java specifications in the file names.

http://docs.oracle.com/javase/7/docs/api/java/util/
PropertyResourceBundle.html

Character encoding must be done according to the Java language specification for those resource properties bundles that will be used for target metadata, jobs, and so on.

Encoding of Flex resource properties files does not follow the same encoding as the Java language specification. Therefore it is necessary to separate any string resources that will be displayed in the Flex UI (MPCUI) into separate resource properties bundles. For more information, see the Flex Documentation.

## Packaging Resource Bundles

To package resource bundles:

1. Add the Resource properties files to a plug-in using a JAR file that includes the plug-in staging area under the oms/archives directory from where the plug-in OPAR is created. This jar file can contain properties files only and not any other files such as Java class files, images, and so on.

2. Include the properties files in a directory where the path is the three part plug-in id, followed by a subpackage name of your choice. For example, if the plug-in id is test.group.domain, then the path to the resource properties files must be test.group.domain.rsc, where rsc is the subpackage selected for the resource bundles as described in Define the Resource Bundle Package.

3. Using the previous example, enter the following JAR command to create a JAR file to include in the oms/archives directory of the plug-in stage area:

   ```
   jar cvf test_group_resources.jar test/group/domain/rsc/*
   ```

4. Place the JAR file in the oms/archives directory and then the EDK tools can validate and package the plug-in. For more information, see Validating, Packaging, and Deploying the Plug-in .

   > **Note:**
   >
   > If the JAR file contains anything other than properties files, then the following validation error appears:
   >
   > ```
   > Plug-ins of type MP and MPP can only contain resource bundles in
   > archives in java properties format. Found other files in artifact.
   >
   > ./stage/oms/archives/test_group_resources.jar
   > ```

> **Note:**
>
> For examples of resource properties bundles, see the following pages from The Java Tutorial:
>
> http://docs.oracle.com/javase/tutorial/i18n/resbundle/propfile.html

# Checking a New Target Type

Before you register a new target type, check the following list:

- Target Type Name

  Ensure that the target type name follows the following naming format, where *plugin* represents the plug-in name and *type* represents the target type name:

  ```
  oracle_plugin_type
  ```

  For example, oracle_vt_zone or oracle_emas_forms_server.

- Model
  - Ensure that the target is a manageable entity or that it can be monitored. Also, ensure that it makes business sense to model the target type.
  - Ensure that the target type has a response metric and other quantifiable numeric metrics
  - Ensure that the target type is a required target and has an identifiable presence even if Enterprise Manager is not installed.
  - Identify the manageable entity class to which the target type belongs and set the property correctly.
  - Ensure that the Response metric has one numeric metric column only called Status

- Version
  - Ensure that the metadata version is defined correctly. For more information about the metadata version number, see Defining the Target Type Metadata.

- Associations
  - Ensure that no abstract association types are used. For example:

    ```
    select assoc_type from mgmt_assoc_types where is_abstract=1
    ```

  - Ensure that core association types are used. For example:

    ```
    select assoc_type from mgmt_assoc_types where category=1
    ```

  - Do not define the provided_by/relies_on_key_component allowed pair. The Service framework automatically adds a service (provided_by/relies_on_key_component allowed pair).

- Properties
  - Do not store credentials such as user names and passwords in the target properties.

- Include properties that are used for monitoring the target *only* in the target properties. If data is not actively used by the Management Agent, then it is not a target property.

- Add a target version property to capture the target version. By default, Enterprise Manager uses a *TargetVersion* property to represent the target version. For more information about this property, see Table 3-2.

# Testing Your Target Type Definitions

Test your new target type definitions by using the metric browser. The metric browser is a development utility that is an integral part of the Management Agent. As a subsystem of the Management Agent, it allows you to quickly access the metric values for targets monitored by the Management Agent without the overhead of a Management Repository or other components of the Enterprise Manager framework.

## Activate the Metric Browser

To configure the Management Agent's metric browser for debugging metrics without the Enterprise Manager Cloud Control console:

1. Check that the `_enableMetricBrowser` line in the $*AGENT_HOME*/sysman/config/ emd.properties file is enabled, where *AGENT_HOME* represents the home directory of the Management Agent:

   ```
   _enableMetricBrowser=True
   ```

2. Enter the following command to apply the changes that you made to the emd.properties file:

   ```
   emctl reload agent
   ```

3. Open the emd.properties file and check the EMD_URL line. It has the following format:

   ```
   EMD_URL=http://host:port/emd/browser/main
   ```

Alternatively, you can use the `emctl` command to activate the metric browser as follows:

```
emctl setproperty agent -name _enableMetricBrowser -value true
```

## View Your Metrics

After the target instance has been added to the targets.xml file and the new information has been reloaded, you can view available targets and metrics through the metric browser. Access the following URL using any web browser:

```
http://host:port/emd/browser/main
```

> 💡 **Tip:**
>
> To find the port number used by the Management Agent, open the $*AGENT_HOME*/sysman/config/emd.properties file and search for the EMD_URL line.

> ✏ **Note:**
>
> You must use the Management Agent operating system credentials to log in to the metric browser.

# Validating Your Metadata XML

To verify that your target metadata files are defined correctly, enter the following command from the bin directory of the EDK:

```
empdk validate_plugin -stage_dir plugin_stage
```

In the preceding command, *plugin_stage* represents the plug-in staging directory.

For information about the EDK, see About the Extensibility Development Kit (EDK) and for more information about the `empdk` command and its usage, see Validating the Plug-in.

# Troubleshooting the Target Creation Process

This section provides some troubleshooting tips if you encounter any issues with your targets.

**My target is not added to Enterprise Manager**

If your target is not added, do the following:

- Check the Oracle Management Service trace file (emoms.trc) for exceptions. The OMS trace file is located in the *EM_INSTANCE_BASE*/em/*OMS_NAME*/ sysman/log/ directory, where *EM_INSTANCE_BASE* is the OMS Instance Base directory (by default, this directory is under the parent directory of the Oracle Middleware Home).

  ```
  grep -i EntityManager.createEntities *
  grep -i EntityUtil *
  ```

- If your target is added to the Management Repository but not to the Management Agent, go to the agentStateDir/sysman/log directory and check the Management Agent log file (gcagent_mdu.log). This log file tracks the metadata updates to the Management Agent.

**My target continues to show a pending status**

If your target is monitored by the Management Agent and it shows a pending status, then do the following:

- Check if the Management Agent is still monitoring the target.

  To list the name and type of each target being monitored by a Management Agent:

  1. Change directory to the *AGENT_HOME*/bin directory (UNIX) or the *AGENT_HOME*\bin directory (Windows).

  2. Enter the following command:

     ```
     emctl config agent listtargets
     ```

**3.** Check the output for your target.

- Check that the plug-in is deployed on the Management Agent by reviewing the following log file:

```
agent_inst/sysman/registry.xml
```

- Check that the Management Agent received the request to add the target. Go to the agentStateDir/sysman/log directory and review the Management Agent log file (gcagent_mdu.log).

- From a SQL*Plus session, run the tgtinfo.sql script, similar to:

```
@tgtinfo oracle_database orcl
```

The tgtinfo.sql script includes the following:

```
SELECT type_meta_ver, ':'||category_prop_1||':'||
                       category_prop_2||':'||
                       category_prop_3||':'||
                       category_prop_4||':'||
                       category_prop_5||':' category_prop,
       target_guid,
       TO_CHAR(load_timestamp,'DD_MON-YY HH24:MI:SS'),

timezone_region,owner,host_name,emd_url,broken_reason,broken_str,manage_status,
       promote_status,
       dynamic_property_status
  FROM sysman.em_targets
 WHERE target_type='&&1'
   AND target_name='&&2'
/
```

> **✎ Note:**
>
> If you are having issues running the script, edit the script to replace `&&1` with the type of the target and replace `&&2` with the name of the target.

The output from the script includes the following information:

- TARGET_TYPE

  Name of the target, such as oracle_database

- TYPE_META_VER

  Metadata version number. Check that the metadata version is correct for the target.

- CATEGORY_PROP_1

  Category properties can be used to distinguish different metric definitions based on different configurations. Check that the value is correct for the target.

- BROKEN_REASON

  If this value is greater than 0, then target is broken (for example, the target could not be saved or it is missing required properties). The BROKEN_STR value will provide a reason as to why the target is broken.

- MANAGE_STATUS

The manage status of the target. Possible values include:

* 0: Ignored

* 1: Not yet managed

* 2: Managed

* 3: Managed target component

– PROMOTE_STATUS

The promotion status of the target. Possible values include:

* 0: Cannot promote (an existence-only entity)

* 1: Eligable for promotion

* 2: Promotion in progress

* 3: Promoted to Management Agent

* 4: Promotion in progress but target was added to the Management Agent previously

– DYNAMIC_PROPERTY_STATUS

Status of the dynamic properties. Possible values include:

* 0: Dynamic properties have not been uploaded by the Management Agent

* 1: Dynamic properties are uploaded by the Management Agent

# 4
# Plug-in Builder

This chapter describes the following topics:

## Overview

The Enterprise Manager Plug-in builder is a JDeveloper extension that helps integrators to create plug-ins using JDeveloper editor. The intuitive GUI wizards available within the plug-in builder enable you to easily develop plug-ins that can be imported and deployed onto Enterprise Manager. Traditional way of creating a Metadata Plug-in using various XML editors has always been prone to semantic and syntactic errors. Therefore, Oracle recommends using this interactive development environment to take advantage of the various run time validation intelligence embedded in the extension.

To develop plug-ins, you need to download the plug-in builder tool that is shipped with Extensibility Development Kit (EDK). To download the EDK kit, from Cloud Control console, select **Setup**, then **Extensibility**, and **Development Kit.** Following are the key components required to develop metadata plug-ins:

- **Plugin.xml:** A `plugin.xml` file provides the metadata describing the plug-in, and is used for deploying plug-ins. It contains properties that identify the plug-in, such as name and version, and declares the set of target types that will be added to Enterprise Manager Cloud Control.

- **Plugin_registry.xml**: A `plugin_registry.xml` file provides the metadata required by the Management Agent to which the plug-in will be deployed. It is packaged in the `/agent` directory within the plug-in archive and is deployed to the Management Agent that will monitor a target.

- **Target Type**: A target type metadata file is an integral part of defining a new target type. The target type file describes a set of metrics that can be collected for a specific type of target. Essentially, it tells the Management Agent what data to retrieve and how to obtain that data for this particular target type. To add a new target type, provide the following details:

    - **Instance properties** defines what properties an administrator must specify in the Enterprise Manager Cloud Control console when adding a new target instance of this particular target type.

    - **Credentials** are required for a plug-in to authenticate with each target instance that it will collect data for, or run jobs against. Credential types and credential sets are needed to enable authentication.

    - **Metrics** are at the core of Cloud Control's target monitoring capabilities. Basically, Cloud Control's ability to monitor and manage various targets. This in-turn refers to its ability to collect, process, and display target metrics.

- **Default Collection:** The default collection file defines the metric data to be collected from targets and written to the Management Repository along with information such as the collection frequency. The default collection metadata file for a target type defines the following:

    - The frequency of at which this metric data is collected.

    - Thresholds that, when exceeded, will cause a Metric Alert event to be raised.

    - An optional message to display when a threshold is exceeded.

# Prerequisites For Using Plug-in Builder

- Ensure that you have downloaded and installed the latest version of JDK 7 on your system.

- For developing plug-ins using plug-in builder, Oracle recommends using Oracle JDeveloper 12.1.3.0.0 Studio Version. To install this JDeveloper version along with the plug-in, follow the steps listed in Installing Plug-in Builder and a New JDeveloper Instance.

> **✎ Note:**
>
> If you select Oracle JDeveloper 12*g* 12.1.3.0.0 - Java Edition - Generic, then you must have an existing JDeveloper instance running. To install JDeveloper, see `http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html`. Following which, you can install plug-in builder using the steps listed in Installing Plug-in Builder into an Existing JDeveloper Instance.

- Ensure that you have downloaded the latest EDK kit to your local system. To do so, follow the steps listed in Installing the Extensibility Development Kit (EDK).

# Installing Plug-in Builder

This section contains the following topics:

- Installing Plug-in Builder and a New JDeveloper Instance

- Installing Plug-in Builder into an Existing JDeveloper Instance

> **Note:**
>
> After installing the Plug-in Builder, to verify if the Plug-in Builder extension has been properly deployed, follow these steps:
>
> 1. On the Oracle JDeveloper page, from **help** menu, select **About.**
>
> 2. In the About Oracle JDeveloper dialog box, select **Extensions.**
>
> 3. In the Extensions Tab, look for:
>
>    Name: `EM Plug-in Builder`
>
>    Identifier: `oracle.em.edk.pluginbuilder`
>
>    Version: `12.1.0.1.0`
>
>    Status: `Loaded`

## Installing Plug-in Builder and a New JDeveloper Instance

To install the JDeveloper studio version, and the plug-in builder components, follow these steps:

1. Download the Generic Studio Edition of Oracle JDeveloper 12.1.3.0.0 (`jdev_suite_121300.jar`).

2. Set the following environmental variables:

```
On Unix:
export JAVA_HOME=/usr/jdk7
export EDK_HOME=/home/SCHARGE/13.2.0.0.0_edk_partner

On Windows:
set JAVA_HOME=C:\Program Files\Java\jdk7
set EDK_HOME=C:\Users\SCHARGE\13.2.0.0.0_edk_partner
```

3. Run the following command to install JDeveloper Studio binary:

```
On Unix:
$EDK_HOME/bin/setup.sh

On Windows:
%EDK_HOME%\bin\setup.bat
```

## Installing Plug-in Builder into an Existing JDeveloper Instance

If you have an existing JDeveloper instance, then you must use JDeveloper update mechanism to install plug-in builder extension. To do so, follow these steps:

1. Set the following environmental variables:

```
On Unix:
export JAVA_HOME=/usr/jdk7

On Windows:
set JAVA_HOME=C:\Program Files\Java\jdk7
```

**ORACLE**

2. Run the command to start the existing JDeveloper instance.

3. On the Oracle JDeveloper page, from **help** menu, select **Check for Updates.** Check for Updates Wizard is displayed.

4. On the Welcome page, click **Next.**

5. On the Source Page, select **Install from Local File.** Click browse, or enter the path to the plug-in builder file:

   On Linux

   `<EDK_INSTALL_DIR>/lib/empluginbuilder.zip`

   On Windows:

   `<EDK_INSTALL_DIR>\lib\empluginbuilder.zip`

   Where, `EDK_INSTALL_DIR` is the directory where EDK is installed.

   Click **Next.**

6. On the Summary page, upgraded extensions, and the new extensions are displayed. Click **Finish.**

   You are then prompted to exit the JDeveloper wizard.

7. To update the JDeveloper reference to EDK, follow these steps:

   a. Start the JDeveloper instance.

   b. On the Oracle JDeveloper page, from **Tools** menu, select **Preferences**.

   c. On the Preferences page, select **EM Plug- in Builder**, and update the location of the EDK.

   > **Note:**
   >
   > If you miss updating the EDK home location, the plug-in project fails with `set the EDK location` error.

# Creating an Enterprise Manager Plug-in Project

To create an Enterprise Manager project, follow these steps:

1. Run the command to start the existing JDeveloper instance.

2. On Oracle JDeveloper page, from **File** menu, select **New**. A New Gallery dialog box is displayed.

3. On the left pane, from the Categories section, select **Enterprise Manager Plugin**, and click **OK** to create a metadata plug-in project.
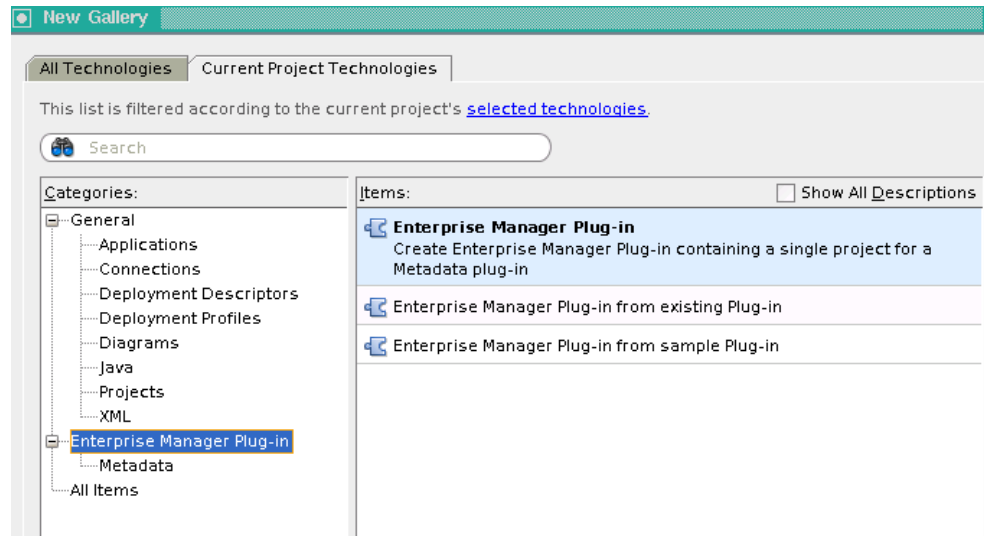
4. In the Create EM Plug-in dialog box, enter the following details:

**Figure 4-1    Create EM Plug-in**
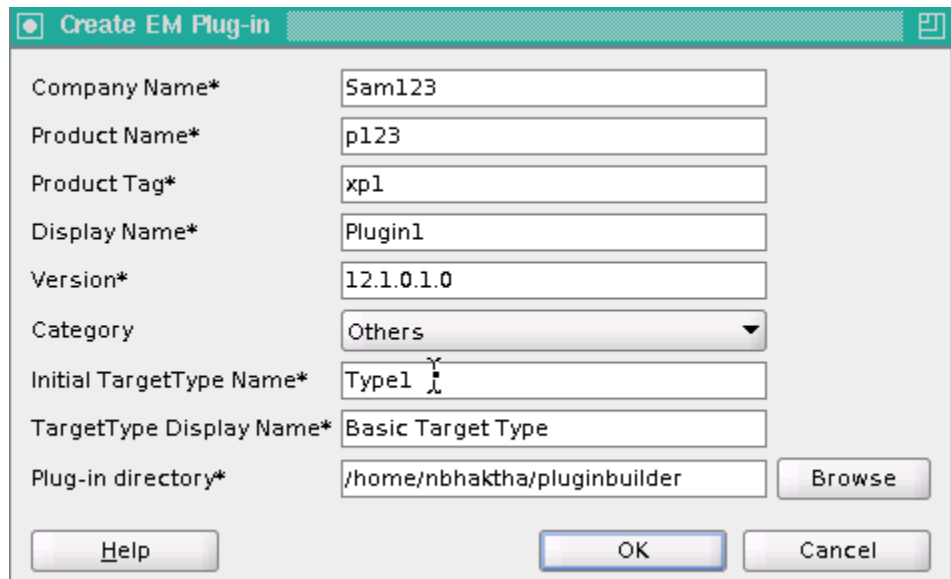


**Table 4-1    Create EM Plug-in**

| Attribute | Description |
| --- | --- |
| Company Name | Ensure that you begin the name with an alphabetic value. The length can range from one to eight characters. For example: Sam123 |
| Product Name | Ensure that you begin the product name with an alphabetic value. The length can range from one and eight characters. For example: p123 |

**Table 4-1    (Cont.) Create EM Plug-in**

| Attribute | Description |
| --- | --- |
| Product Tag | Ensure that you start the product tag with `x`. The length of this field can range from two and four characters. For example: `xp1` |
| Display Name | Descriptive display name for the plug-in. For example: Plugin1 |
| Version | This is a pre-populated value. Describes the version of the plug-in. For example: 12.1.0.1.0. |
| Category | Select a category from the menu. By default it is `Others`. |
| Initial TargetType Name | This is a pre-populated value that contains the suffix for initial target type. For example: Type 1 |
| TargetType Display Name | Descriptive display name for target type. For example: Basic Target Data. |
| Plug-in Directory | This is the location where the plug-in project is created. Click **Browse**, to change the directory location. For example: `/home/nbhaktha/pluginbuilder.` |

Click **OK**.

5. A new project is listed in the Application Navigator tab. If you have created more than one project, then you can select the desired plug-in project from the Application Navigator menu. When you expand the project, you will see three primary resources: agent, discovery, and oms.

6. Right-click **plugin.xml** to view the details of the plug-in. Click **Target Types** tab to view all the targets types added to the current plug-in. Click **Collection Items** tab to view the list of collection items associated with this target.

7. To add new target types to a plug-in, see Adding a New Target Type.

8. To add collection items for a target, see Adding a Collection Item for the Target.

> **Note:**
>
> Starting with Enterprise Manager 12.1.0.4, out-of-box support for discovery of targets is present. For more information, see Discovering Targets.

9. Click refresh icon to view all the targets and collection items added into a plug-in project. Note that these files are not physically added into the agent directory until the staging phase. For information about how these files are packaged into a plug-in project, see Appendix.

10. Right-click the project name and select **Validate Plugin Distribution** from the context menu.

11. In the Validate EM Plugin dialog box, the Plugin Staging directory and Output Directory values are pre-populated, you may change them if required. Click **OK**.

12. Right-click the project name, and select **Create Plugin Archive** from the context menu.

13. In the Create Plugin Archive dialog box, the Plugin Staging Directory and Output Directory values are pre-populated; you may change them if required. Click **OK** to create the `.Opar` file.

   For more about importing and deploying the `.Opar` files, see Deploying the Plug-in Archive into Enterprise Manager.

# Creating a Plug-in Project Using Sample Plug-ins

To create an Enterprise Manager plug-in project using sample plug-ins, follow these steps:

1. Run the command to start the existing JDeveloper instance.

2. On Oracle JDeveloper page, from **File** menu, select **New**. A New Gallery dialog box is displayed.

3. In the Categories section, select **Enterprise Manager Plugin**, then select **Enterprise Manager Plug-in from sample Plug-in.**
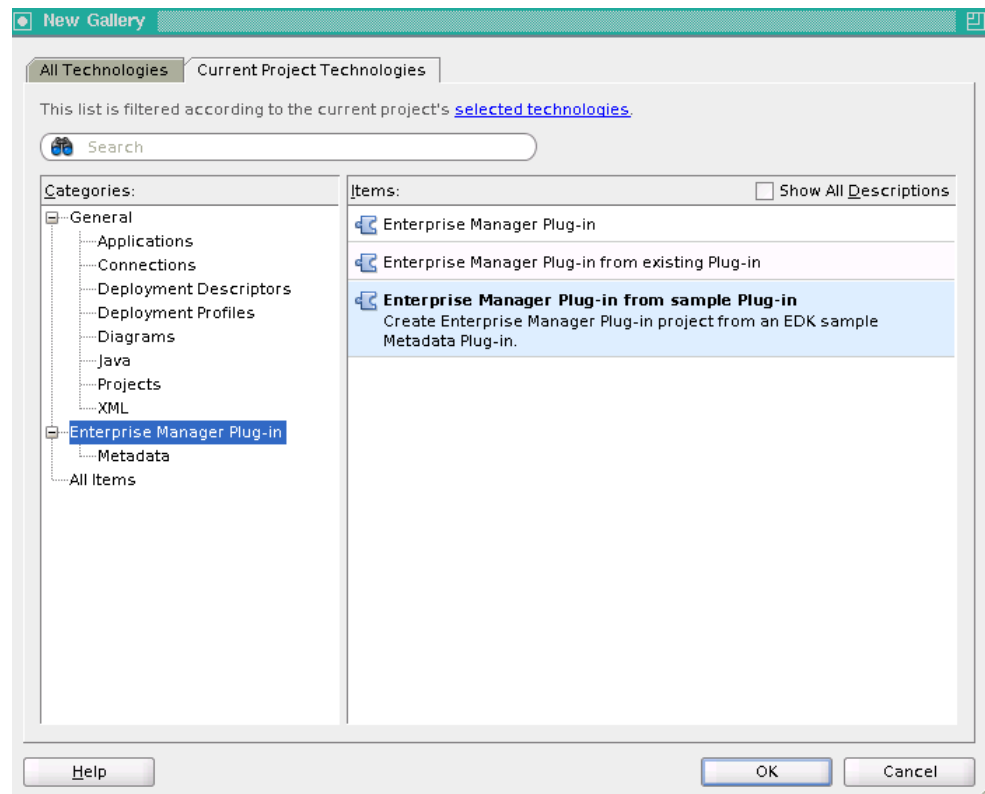


4. Click **OK.**

5. In the Create EM Plugin dialog box, enter all the relevant plug-in details like **Company Name, Product Name, Product Tag**, and **Display Name**.
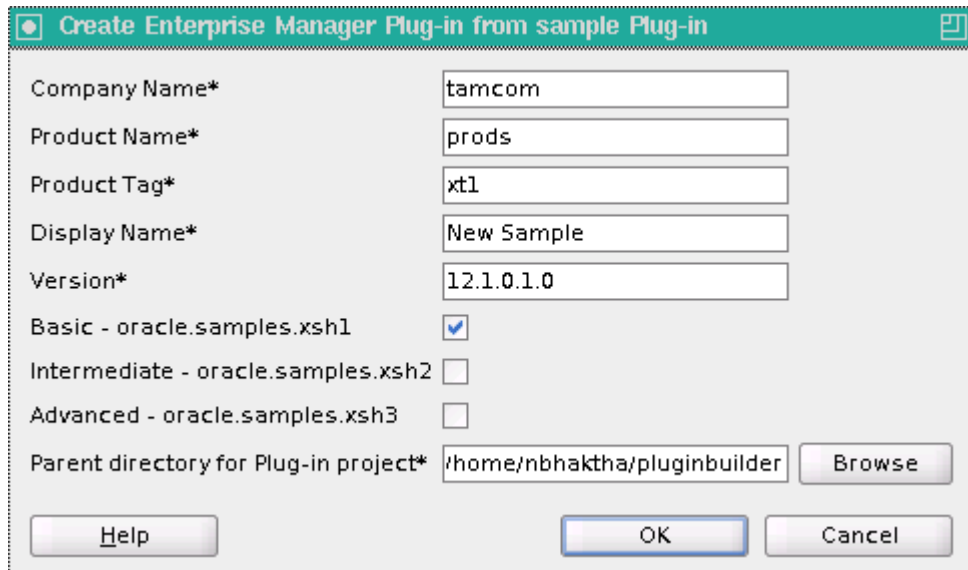
Select one of the following samples from the list:

- `Basic - oracle.samples.xsh1`: This is a basic plug-in demonstrating minimal monitoring capability like Target, Metrics definition and Reports.

- `Intermediate - oracle.samples.xsh2`: This is an intermediate plug-in covering samples of Configuration collections, Job types and Target Associations on top of Basic features

- `Advanced - oracle.samples.xsh3`: This is an advanced plug-in covering Automatic Discovery, BI Publisher reports, Derived Associations and Compliance standards along with Intermediate features

Enter the location for the new plug-in project.

Click **OK**.

This copies the sample into the specified project directory. It then modifies the target type and collection files to use the new standard (`company_product_tag_type`) and corrects the xml contents of the plugin, agent-registry, targettype and default_collection files to reflect the data entered in the wizard.

6. Right-click the project name, and select **Validate Plugin Distribution** from the context menu.

7. In the Validate EM Plugin dialog box, the Plugin Staging directory and Output Directory values are pre-populated, you may change them if required. Click **OK**.

8. Right-click the project name, and select **Create Plugin Archive** from the context menu.

9. In the Create Plugin Archive dialog box, the Plugin Staging Directory and Output Directory values are pre-populated; you may change them if required. Click **OK** to create the `.Opar` file.

For more about importing and deploying the `.Opar` files, see Deploying the Plug-in Archive into Enterprise Manager.

# Discovering Targets

Starting with EDK 12.1.0.4.0, plug-in builder supports specifying discovery metadata and target specific discovery code. That means, whenever a new target type is created, the discovery metadata gets updated automatically to include the newly added target types. If you do not want to discover some target types, then you can manually delete that information from the `discovery.xml` file available at: `<project_name>/Resources/oms/metadata/discovery`. To access and view the details of this xml file, see Viewing Basic Discovery Information. The target specific discovery code is supported using a Perl script located under `/Resources/discovery` folder. For any new target type added, the corresponding discovery code should be added in the Perl script. For an example on how to update the Perl script to discover targets, see Discovery Integration Example Requiring User Input.

> **See Also:**
>
> For more information about how discovery of targets can be accomplished in Enterprise Manager manually, see Defining Target Discovery .

## Updating Discovery Metadata for a Pre-existing Plug-in

If you have a plug-in that was created without the discovery metadata, then you can use the Plug-in Builder to manually add the discovery files to the correct folders. Follow these steps to enable discovery support for a pre-existing plug-in:

1. Select the project name from the Application Navigator menu.

2. Navigate to the following directory:

   `<Project_Name>/Resources/oms/metadata/discovery`

   If the **discovery** folder does not exist, then you will need to add it manually.

3. Edit the `<company_name>_<product_name>_<tag>_discovery.xml` file available in the **discovery** folder to add the target type information to the metadata file. If the metadata file does not exist, you may have to add it manually.

   For details about what to add or edit in the discovery metadata file, refer to the sample file available at: `$EDK_HOME/samples/plugins/oracle.samples.xsh3/plugin_dist/oms/metadata/discovery/sample_host3_discovery.xml`

4. To add the Perl script that contains the logic to discover target types, navigate to the following location:

   `<Project_Name>/Resources/discovery`

   If the **discovery** folder does not exist, then you will need to add it manually.

5. Edit the Perl script `<company_name>_<product_name>_<tag>_discovery.pl` to add the relevant logic to discover all the target types that have been included in the metadata file for discovery. If the Perl file does not exist, you may have to add it manually.

For details about what to add or edit in the Perl file, refer to the sample Perl script available at: `$EDK_HOME/samples/plugins/oracle.samples.xsh3/plugin_dist/discovery/sample_host3_discovery.pl`.

6. Right-click the project name, and select **Validate Plugin Distribution** from the context menu.

7. In the Validate EM Plugin dialog box, the Plugin Staging directory and Output Directory values are pre-populated, you may change them if required. Click **OK**.

8. Right-click the project name, and select **Create Plugin Archive** from the context menu.

9. In the Create Plugin Archive dialog box, the Plugin Staging Directory and Output Directory values are pre-populated; you may change them if required. Click **OK** to create the `.Opar` file.

   For more about importing and deploying the `.Opar` files, see Deploying the Plug-in Archive into Enterprise Manager.

## Viewing Basic Discovery Information

To view the discovery information for all the target types that are bundled into a plug-in, follow these steps:

1. Select the project name from the Application Navigator menu.

2. Navigate to the following directory:

   `<Project_Name>/Resources/oms/metadata/discovery`

3. Double-click the xml file available in this directory to view the details. Essentially, the xml records the following metadata information:

   • General tab contains the details of the perl script that needs to be updated to discover targets when the Oracle Management Agent starts up after Enterprise Manager installation.

   • Target Type Included tab lists of all the target types that are eligible for discovery.

# Deploying the Plug-in Archive into Enterprise Manager

Before you deploy the plug-in archive file into Enterprise Manager, perform the following pre-requisite tasks:

1. The Enterprise Manager instance where you plan to deploy the plug-in, must have the Software Library configured, and contain the plug-in archive file.

2. If you want to deploy the plug-in using the Plug-in Builder, ensure that the required preferences are set. To set the Enterprise Manager Plug-in Builder preferences, from the **Tools** menu, select **Preferences**. In the Preferences dialog box, select **EM Plug-in Builder,** and select **I would like to specify a test Enterprise Manager installation.**

Once the plug-in archive file (*.opar*) is ready, to deploy the plug-in archive file into Enterprise Manager, right-click the project name, and from the context menu, select **Deploy Plug-in**, and then click **OK.**

> **Note:**
>
> If you haven't created the Plug-in archive file, you will not be allowed to perform the deploy step.

After the plug-in has been successfully deployed, to access the plug-in, log into Cloud Control, and from the **Setup** menu, select **Extensibility**, then click **Plug-ins.** You must see the newly deployed plug-in on the Plug-ins page.

# Adding a New Target Type

There are two approaches to add a new target type. They are:

**Table 4-2    Adding A Target Type**

| Approach 1 | Approach 2 |
|---|---|
| On Oracle JDeveloper page, from **File** menu, select **New**. A New Gallery dialog box is displayed. | Select the project name from the Application menu. |
| In the Categories section, select **Metadata,** then select **Target Type.** | Expand the **oms** folder present inside the project folder, and drill down to the **targetType** folder. |
| Click **OK.** | Right-click the targetType folder, and select **New** |
| NA | In the New Gallery window, select **Metadata**., then click **Target Type.** |
|  | Click **Ok.** |

1. In the Add Target Type dialog box, enter a unique name and a display name for the new target type, then click **OK.**
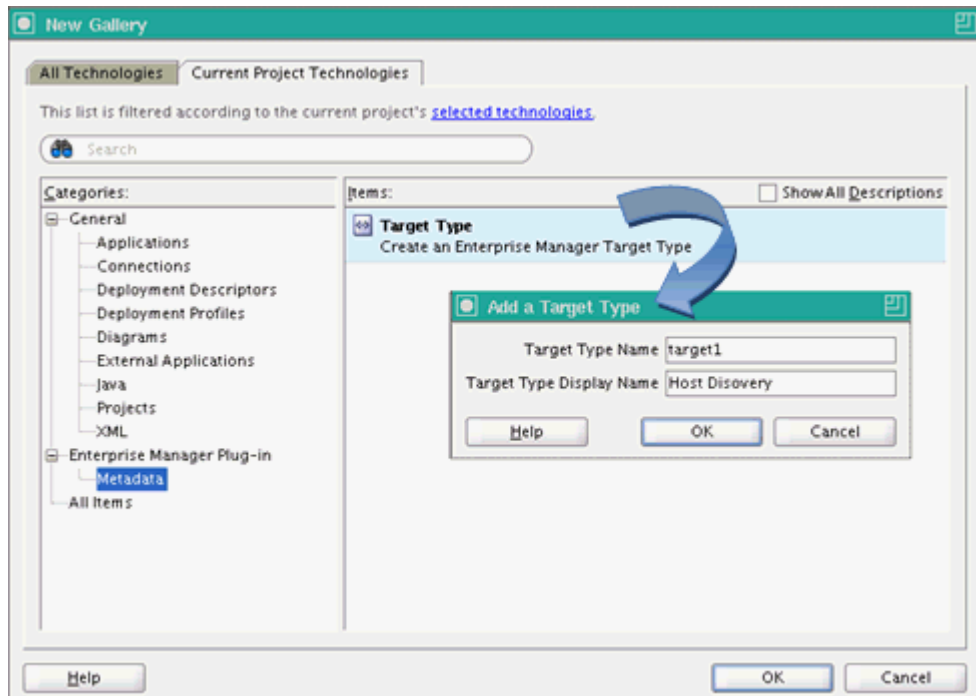
2. Click **Refresh** icon available in the Projects tab to view the new target type added.

Double-click the new target type added to view the details or to update the details. For more information, see Updating Target Type Information.

> **Note:**
>
> The newly added target type automatically becomes available for discovery. In order to discover the targets, you must ensure that the perl script available in `/Resources/discovery` directory is manually updated to include all the target type information. For more information, see Discovering Targets.

# Updating Target Type Information

For a new Target Type, you need to provide the following details:

- Adding Instance Properties
- Adding Dynamic Properties
- Adding Credential Type
- Adding Credential Set
- Adding Metric Properties for a Target

> **Note:**
>
> The metric references are completely synchronized with the collection metadata, which means, if you delete or rename a metric for a target type, then the corresponding reference in the collection items are also updated accordingly.

## Adding Instance Properties

To define what properties an administrator must specify in the Enterprise Manager Cloud Control console when adding a new target instance of this particular target type, follow these steps:

1.  Double-click the target type file (`target_type.xml`) to open with the Overview editor.

2.  Select **Properties** tab, and click **add (+)** in the InstanceProperty section.

3.  In the Insert InstanceProperty dialog box, enter a `Name`, `NLSID`, `LABEL` for the property. By default `OPTIONAL` is set to False (unchecked), which means that a property must be specified.

    For example, you can add a property by name `Password`, and make OPTIONAL as `false` (by deselecting the check box), which implies that the Administrator will need to specify a password while adding a new target instance for this particular target type.

4.  Click **OK.**

    You can edit or delete the property by selecting the respective option available in the InstanceProperty section.

## Adding Dynamic Properties

The values for dynamic instance properties are passed back by the Management Agent collecting data from the target instance. They are typically used within a QueryDescriptor to define properties passed to the fetchlet responsible for metric collection. To add dynamic instance properties, follow these steps:

1.  Double-click the target type file (`target_type.xml`) to open with the Overview editor.

2.  Select **Properties** tab, and click **add (+)** icon in the DynamicProperties section.

3.  In the Insert DynamicProperty dialog box, enter a `Name`, `PROP_LIST`, and select `FORMAT`, and `FetchLet` option from the menu.

    ```
    For example:
    NAME: AruidInfo
    PROP_LIST: ARUID
    FORMAT: ROW
    FetchLet: OSLineToken
    ```

4.  Click add **(+)** icon in the Query Properties section. In the add QueryDescriptor Property dialog box, enter a name and define the scope for the new property.

    ```
    NAME: scriptsDir
    SCOPE: GLOBAL
    ```

5.  Click **OK.**

    You can edit or delete the property by selecting the respective option available in the DynamicProperties section.

## Adding Credential Type

Credential type is the type of authentication supported by a target type. To add a credential type, follow these steps:

1.  Double-click the target type file (`target_type.xml`) to open with the Overview editor.

2.  Select **Credentials** tab, and click **add (+)** in the CredentialType section.

3.  In the Insert CredentialType dialog box, enter a Name, NLSID, and label for the credentials you are adding.

4.  Click **add (+)** icon in the CredentialType Columns section.

5.  In the add Credentialtype Column dialog box, enter the column values for each credential type, and click **OK.**

For example, to create a host credential with two columns UserName and Password, you need to provide the following details:

```
In the Insert CredentialType dialog box, enter the following details:
NAME: XP2HostCreds
NLSID: CREDS_HOST_HOSTCREDS
LABEL: XP2 Host Credentials

In the AddCredentialType Column, enter the following details:
NAME: XP2HostUserName
NLSID: CREDS_HOST_USERNAME
LABEL: XP2 Host UserName

NAME: XP2HostPassword
NLSID: CREDS_HOST_Password
LABEL: XP2 Password
```

You can edit or delete the Credentials by selecting the respective option available in the CredentialType section.

## Adding Credential Set

To create an instance of a CredentialType, follow these steps:

1.  Double-click the target type file (`target_type.xml`) to open with the Overview editor.

2.  Select **Credentials** tab, and click add (**+**) in the CredentialSet section.

3.  In the Insert CredentialSet dialog box, enter a unique name, select the Credential type from the menu, select a value for usage, enter an NLSID, and a label for the instance of the credential type that you are creating.

4.  Click add (**+**) icon in the CredentialTypeColumns section.

5.  In the Add CredentialSet Column dialog box, enter the column values for each credential set, and click **OK**.

For example, to create an instance of Host Credential type called Normal host credential with two columns **Normal Username** and **Normal Password,** you need to provide the following details:

```
In the Insert CredentialSet dialog box, enter the following details:
NAME: HostCredsNormal
CREDENTIALTYPE: XP2HostCreds
USAGE: PREFERRED_CRED
NLSID: CREDS_HOST_HOSTCREDS_NORMAL
LABEL: Normal Host Credentials

In the AddCredentialSet Column, enter the following details:
SET_COLUMN: username
TYPE_COLUMN: XP2HostUsername
NLSID: CREDS_HOST_HOSTCREDS_NORMAL
LABEL: Normal Username

SET_COLUMN: password
TYPE_COLUMN: XP2HostPassword
NLSID: CREDS_HOST_HOSTCREDS_NORMAL
LABEL: Normal Password
```

You can edit or delete the Credentials by selecting the respective option available in the CredentialType section.

## Adding Metric Properties for a Target

To add a metric property, follow these steps:

1. Double-click the target type file (`target_type.xml`) to open with the Overview editor.

2. Select **Metrics** tab, and click add (**+**) in the Metric section. The Metric Properties dialog box is displayed.

3. In the general tab, enter name, NLSID, value, and type for the metric.

4. In the TableDescriptor tab, provide a name for the table, and follow the steps listed in Adding ColumnDescriptor to add a ColumnDescriptor.

5. In the QueryDescriptor tab, select **IncludeQueryDescriptor** if you want to run a query. Select the FETCHLET_ID from the menu, and insert the Query Properties. To add Query Properties, see Adding QueryProperties.

6. In the ExecutionDescriptor tab, enter relevant values in the GetTables and GetViews section. To do so, see Adding ExecutionDescriptor Table and Adding ExecutionDescriptor View.

7. In the Configure Collection tab, to setup data configuration for the metric, click **Add new.** For more information, see Configuring Collection Items for a Target.

8. Click **Finish.**

For example, the following graphic describes how to create a metric group containing metrics that collect CPU performance data:
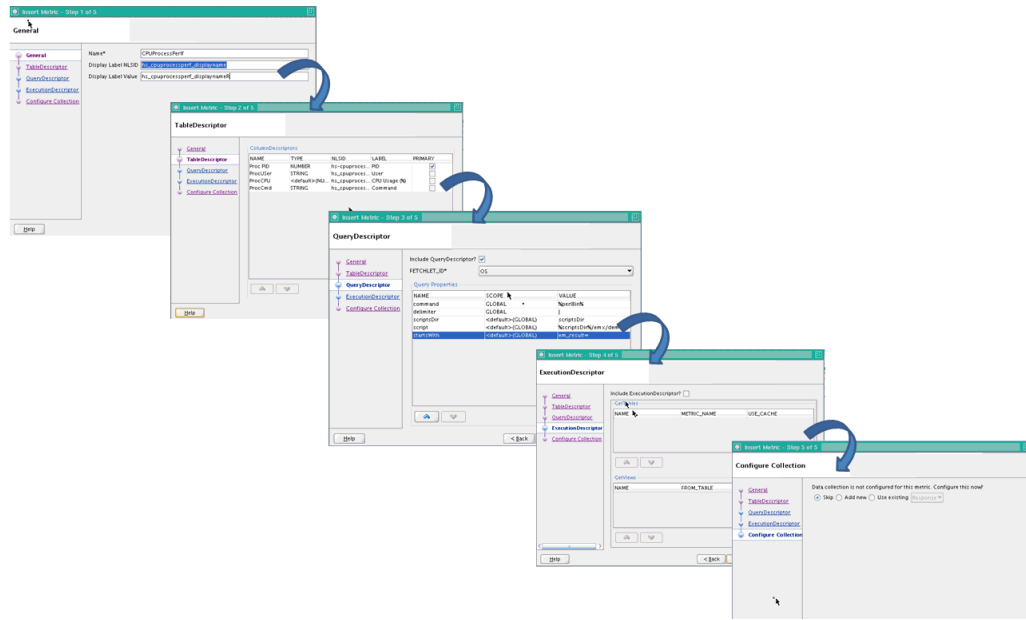
You can edit or delete the metric property by selecting the respective option available in the Metric section.

# Adding ColumnDescriptor

To add a ColumnDescriptor, follow these steps:

1. In the TableDescriptor tab, click add (**+**) icon. Add ColumnDescriptor dialog box is displayed.

2. Enter Name, Type, NLSID, and Label for the output column that you would like to include in the table. Select the IS_PRIMARY check box to make the selected column the primary key column in the Management Repository. Click **OK.**

# Adding QueryProperties

To add a QueryDescriptor, follow these steps:

1. In the QueryDescriptor tab, click add (**+**) icon. Add QueryDescriptor Property dialog box is displayed.

2. Enter Name and Scope for the query property. Click **OK.**

# Adding ExecutionDescriptor Table

To add an ExecutionDescriptor table, follow these steps:

1. In the ExecutionDescriptor tab, click add (**+**) icon in the GetTables section. Add ExecutionDescriptor Table dialog box is displayed.

2. Enter Name and Metric Name, and click **OK.**

# Adding ExecutionDescriptor View

To add an ExecutionDescriptor view, follow these steps:

1. In the ExecutionDescriptor tab, click add (**+**) icon in the GetViews section. Add ExecutionDescriptor Table dialog box is displayed.

2. Enter Name and Metric Name. If you select **Filter**, you can provide a single column name, and a corresponding value. If you do not select **Filter**, click add (**+**) icon to add multiple column names and values. Click **OK.**

## Configuring Collection Items for a Target

In the Configure Collection tab, you can perform the following tasks:

- **Skip:** Choose **skip** to bypass this step. Basically, no collection item is associated with this metric.

- **Add New:** Choose **Add New** to configure a new collection item that will collect data for this metric. To add or edit the properties for a collection item, see Inserting or Updating Collection Item Properties.

- **Use Existing:** Choose **Use Existing**, and select the collection item from the menu to associate an existing collection item to this metric for data collection.

# Adding a Collection Item for the Target

There are two approaches to add new collection item for the target.

**Table 4-3    Adding or Updating a Collection Item**

| Approach 1 | Approach 2 |
|---|---|
| Select the project name from the Application menu. | Select the project name from the Application menu. |
| Expand the **oms** folder for your project, then select **Target Type.**<br>A list of target types is displayed. | Open the plugin.xml file in an overview editor. |
| Select the target type to associate collection item. | Select **Collection Items** tab.<br>A list of all the available collection items for the target is displayed. |
| Select **Metrics** tab, then click Add icon. In the Insert Metric wizard, select **Configure Collection**, then click **Add New.**<br>**Note:** To add or edit the properties for a collection item, see Inserting or Updating Collection Item Properties | Select one collection item from the list, and do the following:<br>In the **General** tab, the metadata version and target type information is displayed.<br>In the **Collection Items** tab, the collection item name is displayed.<br>**Note:** To add or edit the properties for a collection item, see Inserting or Updating Collection Item Properties |

# Inserting or Updating Collection Item Properties

To add or edit the properties of a collection item, follow these steps:

1. In the General tab, enter a name for the collection item. Upload value determines the value following which the data will be written to the repository. For example, an UPLOAD value of 6 implies that every sixth collection of data will be written

to the Management Repository. Provide a value for Interval and the time unit. For example, an interval of 5, and time unit of Min would mean that collection will happen at 5 minutes interval. Click **Next**.

2. In the Conditions tab, you can set a metric alert condition. To do so, click Add (+) in the Conditions section. The Add Conditions dialog box is displayed. Enter all the values for your condition here, and click **OK.** Click **Next.**

3. In the Metric Collection tab, provide the necessary details, and click **Finish.**

The following example represents the CollectionItem entry for the basic Response metric group, which includes the Status metric. It specifies that data for this metric should be collected every 5 minutes, which is the standard collection interval for this type of metric. A condition has been set on the Status metric.:

```
In the General tab, enter the following details:
META_VER: 1.0
TYPE: test_demo_targetType

In the CollectionItem tab, enter the following details:
NAME: Response
UPLOAD: 6
INTERVAL: 5
TIME_UNIT: Min

In the Add Condition Dialog box, enter the following details:
COLUMN_NAME: Status
CRITICAL: 0
OPERATOR: EQ
CLEAR_MESSAGE_NLSID: Response_Status_clearalertmessage
MESSAGE: Failed to connect to database instance: %oraerr%.
```

# Deinstalling Plug-in Builder

To deinstall the plug-in builder, follow these steps:

1. Run the command to stop the existing JDeveloper instance.

2. Navigate to the JDeveloper Instance home:

   On Linux

   `<EDK_INSTALL_DIR>/jdevhome/`

   On Windows:

   `<EDK_INSTALL_DIR>\jdevhome\`

   Where, `EDK_INSTALL_DIR` is the directory where EDK is installed.

3. Run the command to manually delete the following files:

   • `jdeveloper/jdev/extensions/oracle.em.edk.pluginbuilder.jar`

   • `jdeveloper/jdev/extensions/oracle.em.edk.pluginbuilder.help.jar`

4. Run the command to restart the JDeveloper instance.

# Appendix

This section contains the following topics:

## Using the Structure View

Structure view shows the element structure of the XML file and can be used to navigate to specific elements whose attributes will then be displayed in the Property Inspector section.

## Using Property Inspector

Instead of using the Overview section, you can choose to use the plug-in builder specific labels and elements available in the property inspector section to modify the source XML code of the plug-in. This section provides all the plug-in specific attributes that make the editing experience user-friendly

> **Note:**
>
> For a complete list of DTD elements and their usage, see Enterprise Manager DTD .

## Directory Structure for a Plug-in Project

The following is a typical example for a plug-in project. This example describes four targets, and their corresponding collection items. The collection items are essentially meant to collect data from these target types. There are four Collection Items, one for each target. A Perl script is available in the discovery folder. The primary function of this script is to automatically discover all the targets added as a part of this project when the plug-in archive is imported and deployed to Enterprise Manager.
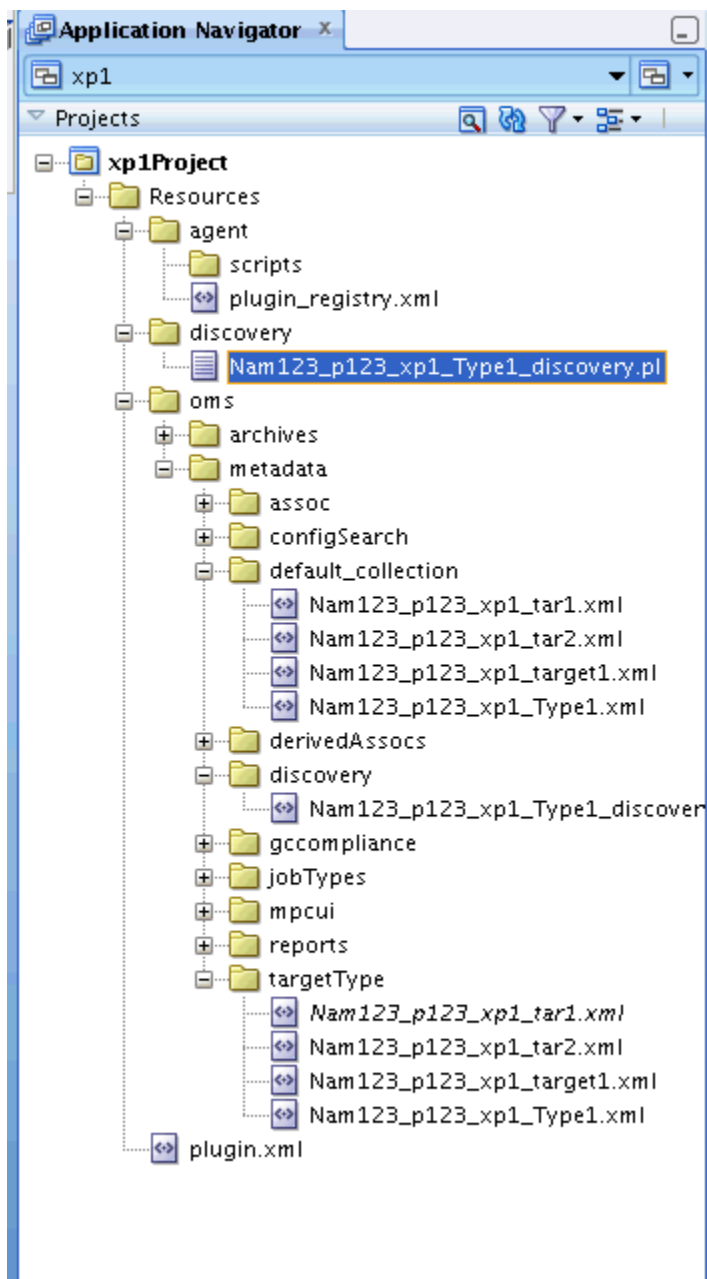
# 5
# Adding Information Publisher Reports

Defining new target types in Enterprise Manager through metadata plug-ins provides you with the opportunity to add new report definitions. Plug-ins also enable you to add permanent (SYSTEM) target type specific report definitions to Enterprise Manager using the Information Publisher XML file format.

> **Note:**
>
> Information Publisher is deprecated as of Enterprise Manager 12*c* and Oracle recommends using BI Publisher to create new reports.

This chapter includes the following sections:

- Introduction to Adding Information Publisher Reports
- Overview of SYSTEM Reports
- Understanding the Report Definition File
- Creating a Report Definition File
- Understanding the XML Report Definition Interface
- Using the ImportExport.xsd File
- About Enterprise Manager Command Line Interface (EM CLI) Verbs
- About Development Guidelines

## Introduction to Adding Information Publisher Reports

As a plug-in developer, to add Information Publisher reports you must design your reports based on the information that you want to show then create your report definition file as follows:

1. Define the SQL and PL/SQL queries used to extract information from the management repository.

   For more information, see Defining SQL or PL/SQL Queries.

2. Create a test report interactively from the Enterprise Manager console.

   For more information, see Creating a Test Report Interactively from the Enterprise Manager Console.

3. Use EM CLI to generate the report definition file.

   For more information, see Using EM CLI to Generate the Report Definition File.

## Assumptions and Prerequisites

This chapter assumes that you are familiar with:

- Management Repository views against which you can write your own queries.

- The XML file format which you will use to create your report definition.

# Overview of SYSTEM Reports

Adding report definitions through metadata plug-ins creates target type specific SYSTEM reports. SYSTEM report definitions are handled differently from definitions created through the Information Publisher user interface. SYSTEM reports are permanent and cannot be deleted or edited by Enterprise Manager administrators. You can add multiple report definitions to a metadata plug-in, thereby enabling you to associate multiple reports with a specific target type.

Adding SYSTEM report definitions using metadata plug-ins and the Information Publisher XML files enables users to access reports from the Enterprise Manager console's Information Publisher Report Definition page.

## About the Report Definitions Page

All report definitions added using metadata plug-ins are available from the Information Publisher's Report Definitions page. As with out-of-box SYSTEM report definitions, those added using metadata plug-ins are organized according to report category and subcategory. SYSTEM report definitions cannot be deleted from the Enterprise Manager console.

# Understanding the Report Definition File

A report definition file is an XML file that contains code to extract relevant information from the Management Repository (using repository views) and the report elements used to format and display that data. The Information Publisher API enables you to specify the report elements and parameters that you normally specify when creating a report definition from the Enterprise Manager console. The fully formed report definition file consists of four basic XML tags and takes on a hierarchical tag structure:

- `<ReportDefinition>`

  Defines report identification parameters as well as encapsulates all report elements used to build the report

- `<ReportElement>`

  Defines the graphical display elements such as tables, charts, or text

- `<ReportElementParameters>`

  Defines specific parameters required by individual report elements

- `<ReportWideParmeters>`

  Defines parameters used by all report elements in the report definition file

# Creating a Report Definition File

As previously mentioned, the content of a report definition file consists of XML tags used to construct a report. You will use both the Enterprise Manager console and EM CLI to develop and generate your report definition file.

Metadata plug-ins enable you to define as many report definition files as required for a particular target type.

## About the Report Definition File Development Process

The process of developing a valid report definition file involves three steps:

1. Defining SQL or PL/SQL Queries
2. Creating a Test Report Interactively from the Enterprise Manager Console
3. Using EM CLI to Generate the Report Definition File

## Defining SQL or PL/SQL Queries

The first step in creating your report definition is to create the SQL or PL/SQL queries used to extract the requisite report information from the Management Repository. Enterprise Manager provides management views with which you can safely extract data from the Management Repository without reading from the base tables. Using repository views protects your queries from changes to the repository schema that may occur in future releases and ensures your SYSTEM report definitions remain functional.

The following query extracts information about blackout history for a target. The query uses the MGMT$BLACKOUT_HISTORY, MGMT$BLACKOUTS, MGMT$TARGET, and MGMT$METRIC_CURRENT repository views.

```
SELECT 'senior mts', count(value) FROM mgmt$metric_current
WHERE metric_column = 'Title' and LOWER(value) LIKE '%senior member%' AND
      target_guid = ??EMIP_BIND_TARGET_GUID??

SELECT bh.created_by "Created by", bh.start_time "Start", bh.end_time "End",
bo.reason "Reason", bo.description "Description"
FROM
MGMT$BLACKOUT_HISTORY bh, MGMT$TARGET tgt, MGMT$BLACKOUTS bo
WHERE tgt.target_name = bh.target_name AND tgt.target_type = bh.target_type
       AND tgt.target_guid = ??EMIP_BIND_TARGET_GUID?? AND bo.blackout_guid =
            bh.blackout_guid
ORDER BY end_time desc

UNION
SELECT 'consulting mts', count(value) FROM mgmt$metric_current
WHERE metric_column = 'Title' and LOWER(value) LIKE '%consulting%' AND
      target_guid = ??EMIP_BIND_TARGET_GUID?? ;
```

When an administrator views a report from the Enterprise Manager console that contains this SQL query string, Information Publisher automatically binds the unique identifier for the selected target to the ??EMIP_BIND_TARGET_GUID?? placeholder in the SQL query string. The documentation for Chart from SQL and Table from SQL parameters provides information about this bind variable placeholder as well as others you can include in your SQL query string.

## Creating a Test Report Interactively from the Enterprise Manager Console

After you have written and tested the SQL or PL/SQL query, you can use the Enterprise Manager console to generate a version of your report interactively using the Chart from SQL and Table from SQL report elements. By using the Information Publisher user interface, you can easily prototype reports without having to create a report definition file and import Plug-in Archive (OPAR) files.

You can also use this method of interactive prototyping to refine your queries and ensure that the data extracted from the Management Repository and how that information is rendered in your report meets your reporting requirements.

## Using EM CLI to Generate the Report Definition File

After you are satisfied with the way your report is being rendered by Information Publisher, you are ready to create the report definition file. To do this, use EM CLI to generate the XML based Report Definition file. The EM CLI `export_report` verb exports the report definition you developed using the Enterprise Manager console (stored in the Management Repository) and generates the XML report definition file. For example:

```
>emcli export_report
      -title="resource report"
      -owner="ADMINISTRATOR_JOE"
      -output_file="$HOME/reports/resource_report.xml"
```

After the report definition file is generated, you must edit the XML file to insert your own plug-in specific information such as product_name, component_name, and oms_version.

The following example shows the content of the report definition file for a report detailing host configuration.

**Example: Host Configuration Report Definition File**

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<ReportDefinition title="Host Performance Overview"
description="Overview of host performance" system_report="0"
category="Sample Host Reports" sub_category=
"Performance Reports" show_navigation="1" generate_context="0"
add_toc="0" product_name="EM" component_name="oracle_hostsample"
is_jit_multi_target="0" target_type="oracle_hostsample"
is_jit_target="1" style="BLAF" oms_version="11.1.0.1.0" xmlns="http://
www.example.com/DataCenter/ReportDefinition">

<ReportElement element_row="1" suppress_render="0"
 element_name_nlsid="IPMSG_USER_CHART_FROM_SQL" header_nlsid="Average
CPU
 Utilization (%)" element_type_nlsid="IPMSG_ANY_TARGET_TYPE"
element_order="0">
      <ReportElementParameters
        parameterName="oracle.sysman.eml.ip.render.elem.
        ChartParamController.chartType" parameterValue="pieChart"/>
      <ReportElementParameters parameterName="oracle.sysman.eml.ip.
```

```
        render.elem.sqlStatement" parameterValue="select column_label,
value
        &quot;CPU Utilization (%)&quot; &#xA;
        from mgmt$metric_current where &#xA;
        target_guid = ??EMIP_BIND_TARGET_GUID??&#xA;
        and metric_name = 'CPUPerf'"/>
      <ReportElementParameters
parameterName="oracle.sysman.eml.ip.render.
        elem.ChartParamController.width" parameterValue="200"/>
   </ReportElement>
 <ReportElement element_row="3" suppress_render="0"
element_name_nlsid="IPMSG_USER_CHART_FROM_SQL" header_nlsid="Memory
Utilization (KB)" element_type_nlsid="IPMSG_ANY_TARGET_TYPE"
element_order="1">
      <ReportElementParameters
parameterName="oracle.sysman.eml.ip.render.
         elem.ChartParamController.legendPosition"
parameterValue="south"/>
      <ReportElementParameters parameterName="oracle.sysman.eml.ip.
        render.elem.ChartParamController.chartType"
parameterValue="barChart"/>
      <ReportElementParameters parameterName="oracle.sysman.eml.ip.
        render.elem.sqlStatement" parameterValue="select column_label,
value
        &quot;Memory Utilization (KB)&quot;&#xA;
        from mgmt$metric_current where &#xA;
        target_guid = ??EMIP_BIND_TARGET_GUID??&#xA;
        and metric_name = 'MemoryPerf'"/>
      <ReportElementParameters
parameterName="oracle.sysman.eml.ip.render.
        elem.ChartParamController.visualOrientation"
parameterValue="horizontal"/>
      <ReportElementParameters parameterName="oracle.sysman.eml.ip.
        render.elem.ChartParamController.width" parameterValue="600"/>
   </ReportElement>
</ReportDefinition>
```

## About the Report Lifecycle: Updating Report Definitions

With the ability to add report definitions to Enterprise Manager comes the responsibility of maintaining and updating the report definitions. Familiarity with the way in which Enterprise Manager handles report definitions will enable you to anticipate system behavior and plan for backwards compatibility.

When report definitions are deployed using metadata plug-ins, Enterprise Manager enables newer versions of the report definitions to be installed. Update and redeploy report definitions which are not valid with a newer version of Enterprise Manager with the new version of the plug-in. Enterprise Manager does not install older versions of a report definition.

Design report definitions, and metadata plug-ins in general, with backwards compatibility in mind. Future versions of report definitions should support previous versions of the target type metadata. Report definition-metadata version incompatibility will be most apparent in the following situations:

- Report definitions included with metadata plug-in version 1 and not included with metadata plug-in version 2 will not disappear when version 2 is deployed.

- If version 1 and version 2 of a metadata plug-in are both deployed on the system, Management Agents will collect data based on the metadata of the version installed on that Agent; some will collect for version 1 metadata and some for version 2 metadata. Only the version 2 report definitions will be installed (appear in the Enterprise Manager console). For this reason, version 2 report definitions must support both versions of the metadata.

# Understanding the XML Report Definition Interface

The Information Publisher XML based report definition file provides an easily editable medium for defining and customizing your Information Publisher reports using simple XML tags.

## About Report Definition Tags

Use the following XML tags to define and manipulate report information when creating report definition files.

- <ReportDefinition>
- <ReportElement>
- <ReportElementParamters>

## <ReportDefinition>

The `<ReportDefinition>` tag is the first XML tag that appears in the report definition file and specifies essential information about your report such as title, description, product name, or Oracle Management Service version. The following example shows the `<ReportDefinition>` tag as defined for a host configuration report.

**<ReportDefinition> Tag for the Host Configuration Report**

```
<ReportDefinition
    title="Host Configuration Overview"
    description="Overview of host configuration" system_report="0"
    category="Sample Host Reports"
    sub_category="Configuration Reports"
    show_navigation="1"
    generate_context="0"
    add_toc="0"
    product_name="EM"
    component_name="oracle_hostsample"
    is_jit_multi_target="0"
    target_type="oracle_hostsample"
    is_jit_target="1"
    style="BLAF"
    oms_version="11.1.0.1.0"
    xmlns="http://www.example.com/DataCenter/ReportDefinition">
```

**Tag Attributes**

**Table 5-1    Tag Attributes for the Host Configuration Report**

| Attribute | Description |
|---|---|
| title | Report title. |
| description | Description. |
| category | Category name. |
| sub_category | Subcategory name. |
| target_type | Target type for late binding, or null if not late binding |
| add_hoc | 1=show 0=hide table of contents |
| show_navigation | Show navigation headers in report (tabs, etc) 1=show, 0=hide |
| product_name | Product name, 'EM' (default) |
| component_name | Product name. This must be set to the metadata plug-in target type. |
| oms_version | Version '11.1' (default). |

**Report-wide Parameters**

```
<ReportWideParameters

parameterName="oracle.sysman.eml.ip.render.elem.TimePeriodParam"
          parameterValue="0:1"/>
```

# &lt;ReportElement&gt;

The `<ReportElement>` tag is used to add a new report element to an existing report definition.

**Input**

**Table 5-2    &lt;ReportElement&gt; Tag**

| Parameter | Description |
|---|---|
| element_type_nlsid | The element type name |
| header_nlsid | The element header or null |
| element_order | The order of this element, 1 based |
| element_row | The row for this element, 1 based |

# &lt;ReportElementParamters&gt;

The `<ReportElementParamters>` tag is used to declare the parameters used for a report element. Include all of the report element parameters you want to declare within the `<ReportElement>` tag.

**Table 5-3    &lt;ReportElementParameters&gt; Tag**

| Parameter | Description |
|---|---|
| parameterName | The parameter name |

**Table 5-3    (Cont.) <ReportElementParameters> Tag**

| Parameter | Description |
| --- | --- |
| parameterValue | The parameter value |

# Using Element Parameters

Parameters used by some report elements dictate the operational behavior of those elements. Use the <ReportElementParamters> tag to declare element parameters associated with a <ReportElement>. The parameter names and values for each element type are described in this section.

This section lists the parameters associated with specific report elements.

# About Table Element Parameters

Use the Table element to show a tabular view of query results. The queries must be made against management views.

To declare a Table element, use the following in the <ReportElement> tag:

```
<ReportElementParameters
element_name_nlsid=" IPMSG_USER_TABLE_FROM_SQL"
element_type_ nlsid="IPMSG_ANY_TARGET_TYPE">
```

- Element Name nlsid: IPMSG_USER_TABLE_FROM_SQL

- Element Type nlsid: IPMSG_ANY_TARGET_TYPE

**Time Period**

**Table 5-4    Table Element Parameters Time Period**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TimePeriodParam". |
| Required | No. |
| Default Value | Null. |
| Valid Values | "0:0" for last 24 Hours. |
|  | "0:1" for last 7 Days. |
|  | "0:2" for last 31 Days. |
| Summary | Encoded time period. |

**Sort Column**

**Table 5-5    Table Render Sort Column**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.initialSortColumn". |
| Required | No. |
| Default Value | The first column in result set. |

**Table 5-5    (Cont.) Table Render Sort Column**

| Attribute | Description |
| --- | --- |
| Valid Values | Any valid column name. |
| Summary | If this parameter is set, the sort column indicator is shown for the column with this column name. If not set, the sort column indicator is shown on the first column. In the SQL query, include an 'order by' clause that sorts by this column. |

**Sort Order**

**Table 5-6    Table Render Sort Order**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.initialSortOrder". |
| Required | No. |
| Default Value | "ascending". |
| Valid Values | "ascending" or "descending". |
| Summary | If this parameter is set, the sort column indicator is shown either as ascending or descending, according to the value. If not set, the sort column indicator is shown as ascending. |

**Name Value Pair Display**

**Table 5-7    Name Value Pair Display**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.nameValueDisplay". |
| Required | No. |
| Default Value | <none>. |
| Valid Values | Positive integer value. |
| Summary | If this parameter is set and only one row is returned from the query, the results are displayed in a vertical list of name-value pairs. Set the value of this attribute to the number of name/value columns to be displayed, normally "1". |

**Number of Rows to Show**

**Table 5-8    Number of Rows**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.numRowsToShow". |
| Required | No. |
| Default Value | "10". |
| Valid Values | Positive integer value. |

**Table 5-8    (Cont.) Number of Rows**

| Attribute | Description |
|-----------|-------------|
| Summary | The number of rows to display at one time in the generated table. You can scroll through additional rows using the UI controls. |

**Is PL/SQL Statement**

**Table 5-9    Is PL/SQL Statement**

| Attribute | Description |
|-----------|-------------|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.sqlStatementIsPlSql". |
| Required | No. |
| Default Value | "false". |
| Valid Values | "true" or "false" . |
| Summary | Whether a SQL statement is PL/SQL. |

**SQL or PL/SQL Statement**

**Table 5-10    SQL or PL/SQL Statement**

| Attribute | Description |
|-----------|-------------|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.sqlStatementIsPlSql". |
| Required | No. |
| Default Value | &lt;None&gt;. |
| Valid Values | Any valid SQL SELECT statement. |
| Summary | SQL statements can optionally bind values for targets, locale information, and start/end date. The format of the SQL statement should include bind variable placeholders for the options to be bound.<br><br>Bind Placeholders:<br>• ??EMIP_BIND_RESULTS_CURSOR??<br>    For use with PL/SQL statements to bind a return cursor containing results for display.<br>• ??EMIP_BIND_TARGET_GUID??<br>    For use with SQL or PL/SQL to bind a target GUID.<br>• ??EMIP_BIND_START_DATE??<br>    For use with SQL or PL/SQL to bind a start date.<br>• ??EMIP_BIND_END_DATE??<br>    For use with SQL or PL/SQL to bind an end date.<br>• ??EMIP_BIND_TIMEZONE_REGION??<br>    For use with SQL or PL/SQL to bind a time zone region.<br>• ??EMIP_BIND_LOCALE_COUNTRY??<br>    For use with SQL or PL/SQL to bind a locale country.<br>• ??EMIP_BIND_LOCALE_LANGUAGE??<br>    For use with SQL or PL/SQL to bind a locale language.<br>Do not append a semi-colon (;) to the end of the SQL statement unless it is a PL/SQL statement. |

ORACLE®

**Example: Specifying an Anonymous PL/SQL Block as a Parameter to an Element Definition**

To avoid issues with formatting, generate the report and export it to XML using the EM CLI.

```
<ReportElementParameters
parameterName="oracle.sysman.eml.ip.render.elem.sqlStatement"
parameterValue="BEGIN DECLARE&#xA;&#xA; BEGIN&#xA;&#xA;
open ??EMIP_BIND_RESULTS_CURSOR?? for select &#xA;
bh.created_by &quot;Created by&quot;, &#xA; bh.start_time
&quot;Start&quot;,&#xA; bh.end_time &quot;End&quot;,&#xA; bo.reason
&quot;Reason&quot;,&#xA; bo.description &quot;Description&quot;&#xA; from &#xA;
MGMT$BLACKOUT_HISTORY bh, &#xA; MGMT$TARGET tgt,&#xA; MGMT$BLACKOUTS bo&#xA;
where tgt.target_name = bh.target_name&#xA; and tgt.target_type =
bh.target_type&#xA; and tgt.target_guid = ??EMIP_BIND_TARGET_GUID??&#xA; and
bo.blackout_guid = bh.blackout_guid&#xA;
order by end_time desc;&#xA;&#xA;&#xA; END;&#xA;END;"/>
```

**Named SQL Statement**

**Table 5-11    Named SQL Statement**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.NamedSqlStatement". |
| Required | No. |
| Default Value | <none>. |
| Valid Values | Any valid statement name. |
| Summary | As an alternative to the "oracle.sysman.eml.ip.render.elem.sqlStatement". |
| | You may use a Named SQL statement which refers to an actual SQL statement stored in the Enterprise Manager repository. |
| | You can register a Named SQL statement by providing an XML file containing the name of the SQL statement as well as the SQL query as part of plug in metadata. |
| | For information about the Named SQL XML file XSD definition, see Using the ImportExport.xsd File. |

**Maximum Number of Rows**

**Table 5-12    Number of Rows**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.maxNumberOfRowsAllowed". |
| Required | No. |
| Default Value | "2000". |
| Valid Values | Any scaler numeric value. |
| Summary | Set the maximum number of rows retrieved for display in the table. For example, to show the top 10 xyz's elements, set the value to "10". |

**Null Data String Substitute**

**Table 5-13    Null Data String Substitute**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.nullDataStringSubstitue". |
| Required | No. |
| Default Value | "" |
| Valid Values | A string. |
| Summary | A string that will be substituted for null values returned. |

**Split Table into Multiple Tables by Column**

**Table 5-14    Split Table**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.nullDataStringSubstitue". |
| Parameter String | "oracle.sysman.eml.ip.render.elem.TableRender.tableSplitColumn". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Any valid column name. |
| Summary | If this parameter is set, the table is split into separate tables with subheaders as the value in this column changes. Order the data by this column. |

**Column Group Header**

**Table 5-15    Column Group Header**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.columnGroupHeader"n. |
| Required | No. |
| Default Value | Null. |
| Valid Values | Header string to use for a column group. |
| Summary | This parameter provides a column header string. |
| | This column group header spans columns between the columns specified in "oracle.sysman.eml.ip.render.elem.TableRender.columnGroupStart Col"n and oracle.sysman.eml.ip.render.elem.TableRender.columnGroupEndCol"n. The n suffix is a numeric value starting with 1 for the first column group, sequentially ascending for subsequent column groups. |

**Column Group Start Column**

**Table 5-16    Column Group Start Column**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.columnGroupStartCol"n. |

**Table 5-16    (Cont.) Column Group Start Column**

| Attribute | Description |
| --- | --- |
| Required | No. |
| Default Value | Null. |
| Valid Values | Any valid column name. |
| Summary | Specifies the first column for a given column group. The n suffix is a numeric value starting with 1 for the first column group, sequentially ascending for subsequent column groups. |

**Column Group End Column**

**Table 5-17    Column Group End Column**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.columnGroupEndCol"n. |
| Required | No. |
| Default Value | Null. |
| Valid Values | Any valid column name. |
| Summary | Specifies the first column for a given column group. The n suffix is a numeric value starting with 1 for the first column group, sequentially ascending for subsequent column groups. |

**Use Separate Rows for Values in a Cell**

**Table 5-18    Use Separate Rows for Values Within a Cell**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.useSeparateRowsColumns". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Comma separated list of valid column names. |
| Summary | If this parameter is set, the delimited values of the column with the given name specified will be displayed on separate rows within a containing row cell. More than one column can be designated for this treatment by adding comma-separated column names. |

**Use Separate Rows as Delimiters**

**Table 5-19    Use Separate Rows as Delimiters**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.useSeparateRowsDelimiter". |
| Required | No. |

**Table 5-19    (Cont.) Use Separate Rows as Delimiters**

| Attribute | Description |
|---|---|
| Default Value | , (comma). |
| Valid Values | Any string. |
| Summary | A character used to delimit tokens within a string. |

**Severity Icon in Column**

**Table 5-20    Severity Icon in Column**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.severityColumn". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Any valid column names. |
| Summary | A severity icon is substituted for valid severity values returned. To omit an icon, your result set can contain null values in this column. |

**Availability Status Icon in Column**

**Table 5-21    Availability Status Icon in Column**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.availabilityStatusColumn". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Any valid column names. |
| Summary | An availability status icon will be substituted for valid values returned. To omit an icon your result set can contains null values in this column. |

**Render Image in Column**

**Table 5-22    Render Image in Column**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.imageFilenameColumns". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Comma separated list of column names. |
| Summary | Optional parameter to display the given image filename in the indicated columns. Indicate for which columns the given image should be rendered. Specify a comma separated list of column names. The image filename returned should contain a relative path starting with '/images',for example '/images/xyz.gif'. Normally, a SQL decode function is used to translate a numeric value into the appropriate image filename. |

**Target Type Column**

**Table 5-23    Target Type Column**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.targetTypeColumns". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Comma separated list of column names. |
| Summary | Optional parameter to indicate for which columns the value returned should be used as an internal target type to be translated into a display string for that type. Specify a comma separated list of column names. |

## About Filter Elements

The following table elements are used to create search filters that enable users to filter on rows for multiple table columns. Three different filter types are permitted:

- Text-value
- List of values obtained from a SQL query
- List of values obtained from a comma-separated list in the element definition

**Define Filter Name**

**Table 5-24    Define Filter Name**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterNames". |
| Required | Yes. |
| Default Value | Null. |
| Valid Values | Comma separated list of filter names. |
| Summary | Defines filter names in a comma-separated list. This parameter also defines the ordering of filter elements. |

**Define Filter Prompt**

**Table 5-25    Define Filter Prompt**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterPrompt<name>Filter". |
| Required | Yes. |
| Default Value | Null. |
| Valid Values | CF. |
| Summary | Defines the prompt used in the Reports page for the filter name. The filter value is accessed from the report element's SQL statement through ??EMIP_BIND_PARAM<name>??. Without any other filter-related parameters, this defines a filter which allows the user to provide a value via a text input field. |

**SQL Filter**

**Table 5-26    SQL Filter**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterSql<name>". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Any valid SQL SELECT statement. |
| Summary | Defines the SQL query used to populate a list of values for a filter name that is presented in the UI as a drill-down menu instead of the text input field. |

**List of Filter Names**

**Table 5-27    List of Filter Names**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterList<name>". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Comma separated list of values. |
| Summary | Defines a list of values for a filter name which is displayed in the UI as a drill-down menu. |

**Translate List of Filter Names**

**Table 5-28    Translate List of Filter Names**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterTranslateValues<name>". |
| Required | No. |
| Default Value | No. |
| Valid Values | yes or no. |
| Summary | Defines whether the values provided by filterSql or filterList should be translated to the client locale. |

**Filter Tip Text**

**Table 5-29    Filter Tip Text**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterTip<name>Filter". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Alpha numeric text. |

**Table 5-29    (Cont.) Filter Tip Text**

| Attribute | Description |
| --- | --- |
| Summary | Defines the text for a tool tip shown if the user moves the mouse over the filter UI elements. |

**Default Filter Name**

**Table 5-30    Default Filter Name**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterDefault<name>". |
| Required | No. |
| Default Value | %. |
| Valid Values | Alpha numeric text string. |
| Summary | Defines a default value for filter name. If no default value is given, '%' is used instead. |

**Null Default Filter Name**

**Table 5-31    Null Default Filter Name**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterDefaultsToNull<name>". |
| Required | No. |
| Default Value | Null. |
| Valid Values | yes or no. |
| Summary | When defined, the default value is NULL instead of '%'. |

**Global Filter Elements**

The following parameters act globally on the filter system.

**Display Empty Table**

**Table 5-32    Display Empty Table**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterStartEmpty". |
| Required | No. |
| Valid Values | yes or no. |
| Summary | If the value of this parameter is 'yes', then the report initially displays an empty table. The table is populated when the user clicks the filter button in the UI. |

**Empty Table Headers**

**Table 5-33    Empty Table Headers**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterEmptyTableHeaders". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Comma-separated list of table headers. |
| Summary | Defines the table headers used when starting with an empty table. |

**Table Header Type**

**Table 5-34    Table Header Type**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterEmptyTableHeaderTypes". |
| Required | No. |
| Default Value | VARCHAR. |
| Valid Values | Comma-separated list of table headers. |
| Summary | This defines the table header types (column types) used when starting with an empty table. This is a comma-separated list. If no header types are specified, the table header types default to VARCHAR. |

**Overwrite Table Header Text**

**Table 5-35    Overwrite Table Header Text**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterHeaderText". |
| Required | No. |
| Default Value | Search Filter. |
| Valid Values | Comma separated list of column names. |
| Summary | Overwrites the default filter section header text. |

**Overwrite Default Filter Description**

**Table 5-36    Overwrite Default Filter Description**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterDescriptionText". |
| Required | No. |
| Default Value | Enter values to filter what is shown in the table. |
| Valid Values | Alpha numeric text string. |
| Summary | Overwrites the default filter section header text. |

**Overwrite Default Filter Tip Text**

**Table 5-37    Overwrite Default Filter Tip Text**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterTipText". |
| Required | No. |
| Default Value | The search filter is case sensitive. Use '%' as a wildcard. |
| Valid Values | Alpha numeric text string. |
| Summary | Overwrites the default filter section tip text. |

**Overwrite Default Button Text**

**Table 5-38    Overwrite Default Button Text**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterButtonText". |
| Required | No. |
| Default Value | OK. |
| Valid Values | Alpha numeric text string. |
| Summary | Overwrites the default filter button text. |

**Empty Table Text**

**Table 5-39    Empty Table Text**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.filterEmptyTableText". |
| Required | No. |
| Default Value | (No rows returned). |
| Valid Values | Alpha numeric text string. |
| Summary | Specifies the text to be shown in an empty table before the filter is run. |

## Using Hyperlinks Within Tables

The following parameters are used to implement hyperlinks within tables and incorporate improved link navigation between master and detail views. This method is an alternative to using oracle.sysman.eml.ip.render.elem.TableRender.columnDestReportTitle<num>, which first takes the user to the target selector page.

**Link to Report**

**Table 5-40    Link to Report**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.columnDestHomepageReportTitle<num>". |
| Required | No. |
| Default Value | Null. |
| Valid Values | Report definition link. |
| Summary | This is the same as oracle.sysman.eml.ip.render.elem.TableRender.columnDestReportTitle<num> except that a link to a report definition on the target homepage is created. |

**Display Number of Columns**

**Table 5-41    Display Number of Columns**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.numberOfColumnsShowed". |
| Required | No. |
| Default Value | Number of columns in the SQL. |
| Valid Values | Number. |
| Summary | Defines the number of columns from the element SQL to be displayed in the UI. Additional columns from the SQL query are hidden but can be used to create the hyperlinks to expose data in the detail report. |

**Display Target Name**

**Table 5-42    Display Target Name**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.columnDestTargetIndex<num>". |
| Required | No. |
| Default Value | Null. |
| Summary | Specifies the column (which may be hidden) that contains the target name. The target name is used in the link to populate the target selection on late binding reports. |

**Display Target Type**

**Table 5-43    Display Target Type**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.columnDestTypeIndex<num>". |

**Table 5-43    (Cont.) Display Target Type**

| Attribute | Description |
| --- | --- |
| Required | No. |
| Default Value | Null. |
| Summary | Specifies the column (which may be hidden) that contains the target type. The target type is used in the link to populate the target selection on late binding reports. |

**Display URL**

**Table 5-44    Display URL**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TableRender.columnDestURLIndex<num>". |
| Required | No. |
| Default Value | Null. |
| Summary | Specifies the column (which may be hidden) that contains an arbitrary URL for a given table element. |

**Example: Report definition defining a master report that enables you to drill down using a link to a detail report**

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<ReportDefinition title="My Master Report" description=
"A master report to show master/detail" system_report="0"
category="Test Reports" sub_category="Master and Detail"
show_navigation="1" generate_context="0" add_toc="1"
product_name="EM" component_name="SAMPLE" is_jit_multi_target="0"
is_jit_target="0" style="BLAF" oms_version="11.2.0.1.0" xmlns="http://
www.example.com/DataCenter/ReportDefinition">

<ReportElement element_row="1" suppress_render="0"
 element_name_nlsid="IPMSG_USER_ TABLE_FROM_SQL" header_nlsid="My Master Report
 Table" element_type_nlsid="IPMSG_ANY_TARGET_TYPE" element_order="0">
   <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.
    elem.TableRender.filterEmptyTableHeaders" parameterValue=
    "Target Name, Target Type"/>
   <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
     TableRender.columnDestParamColumnIndexes1" parameterValue="0,1"/>
   <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
    TableRender.filterHeaderText" parameterValue="My Filter Header"/>
   <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
    TableRender.filterDescriptionText" parameterValue="My Filter description"/>
   <ReportElementParameters parameterName="oracle.sysman.eml.ip.
    render.elem.sqlStatement" parameterValue="SELECT TARGET_NAME &quot;Target
    Name&quot;, TARGET_TYPE &quot;Target Type&quot;
    FROM MGMT$TARGET WHERE TARGET_NAME LIKE
    ??EMIP_BIND_PARAMNAME?? AND TARGET_TYPE LIKE
    ??EMIP_BIND_PARAMTYPE??"/>
   <ReportElementParameters  parameterName="oracle.sysman.eml.ip.render.elem.
    TableRender.filterPromptNAME" parameterValue="Name"/>
   <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.
```

```
    elem.TableRender.filterSqlTYPE" parameterValue=
    "select distinct target_type from mgmt$target"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.filterStartEmpty" parameterValue="yes"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.filterTipTYPE" parameterValue="Filter on the target types"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.filterPromptTYPE" parameterValue="Target Type"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.filterTipText" parameterValue="My Tip Text"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.filterNames" parameterValue="NAME,TYPE"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.numberOfColumnsShowed" parameterValue="2"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.columnDestReportTitle1" parameterValue="My Detail Report"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.filterTipNAME" parameterValue="Filter on the target names"/>
  <ReportElementParameters parameterName="oracle.sysman.eml.ip.render.elem.
   TableRender.filterButtonText" parameterValue="My button text"/>
  </ReportElement>
</ReportDefinition>

<?xml version = '1.0' encoding = 'UTF-8'?>
 <ReportDefinition title="My Detail Report" system_report="0"
  category="Test Reports" sub_category="Master and Detail" show_navigation="1"
  generate_context="0" add_toc="0" product_name="EM"
  is_jit_multi_target="0" is_jit_target="0" style="BLAF"
  oms_version="11.2.0.1.0"
  xmlns="http://www.example.com/DataCenter/ReportDefinition">
  <ReportElement element_row="1" suppress_render="0"
    element_name_nlsid="IPMSG_USER_TABLE_FROM_SQL" element_type_nlsid="IPMSG_ANY_
    TARGET_TYPE" element_order="0">
     <ReportElementParameters
       parameterName="oracle.sysman.eml.ip.render.elem.headerParam"
       parameterValue="Target Detail"/>
     <ReportElementParameters
       parameterName="oracle.sysman.eml.ip.render.elem.sqlStatement"
       parameterValue="SELECT TARGET_NAME &quot;Target Name&quot;,
       TYPE_VERSION &quot;Version&quot;
       FROM MGMT$TARGET
       WHERE TARGET_TYPE LIKE ??EMIP_BIND_PARAM2??
       AND TARGET_NAME LIKE ??EMIP_BIND_PARAM1??"/>
  </ReportElement>
</ReportDefinition>
```

## About the Chart Element

The Chart element is used to show a graphical view of query results. The queries must be made against Management Repository views.

- Element Name: IPMSG_USER_CHART_FROM_SQL

- Element Type: IPMSG_ANY_TARGET_TYPE

**Chart Type**

**Table 5-45    Chart Type**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.chartType". |
| Required | No. |
| Default Value | "pie chart". |
| Valid Values | "barChart", "lineChart", "pieChart", "timeSeriesChart", and "timeSeriesBarChart". |
| Summary | Chart type to display. |

**Time Period**

**Table 5-46    Time Period**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TimePeriodParam". |
| Required | No. |
| Default Value | Null. |
| Valid Values | "0:0" for last 24 Hours."0:1" for last 7 Days."0:2" for last 31 Days. |
| Summary | Encoded time period. |

**Fill**

**Table 5-47    Fill**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.fill". |
| Required | No. |
| Default Value | "none". |
| Valid Values | "none", "absolute", or "cumulative". |
| Summary | Indicates if a line chart should fill the area under the lines. "none": no fill under lines."absolute": lines are identical to the "none" setting but with the area under the lines filled."cumulative": causes the values for the lines to be added or stacked, then the areas underneath the lines are filled.Use caution when using the fill attribute to ensure there is no confusion for the report user as to whether the data in the chart is cumulative or absolute. |

**Height**

**Table 5-48    Height**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.height". |
| Required | No. |
| Default Value | "200". |

**Table 5-48    (Cont.) Height**

| Attribute | Description |
| --- | --- |
| Valid Values | *n*, where *n* is any string that will correctly parse to a positive integer. |
| Summary | Sets the display height of the chart in pixels. |

**Horizontal or Vertical**

**Table 5-49    Horizontal or Vertical**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.visualOrientation". |
| Required | No. |
| Default Value | "horizontal". |
| Valid Values | "horizontal" or "vertical". |
| Summary | Visual orientation of the chart. This attribute is only valid with the chartType attribute set to barChart or timeSeriesChart. The attribute does not affect the pieChart. |

**Legend Position**

**Table 5-50    Legend Position**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.legendPosition". |
| Required | No. |
| Default Value | "east". |
| Valid Values | "default", "east", "south". |
| Summary | Specifies where the legend should be placed relative to the chart. |

**Is PL/SQL Statement**

**Table 5-51    Is PL/SQL Statement**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.sqlStatementIsPlSql". |
| Required | No. |
| Default Value | "false". |
| Valid Values | "true" or "false". |
| Summary | Set to "true" to indicate that the SQL statement is a PL/SQL statement. |

**SQL or PL/SQL Statement**

**Table 5-52    SQL or PL/SQL Statement**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.sqlStatement". |
| Required | No. |
| Default Value | <none>. |
| Valid Values | Any valid SQL SELECT statement or PL/SQL block. |
| Summary | The SQL or PL/SQL statement can optionally bind values for targets, locale information, and start/end date. The format of the statement should include a bind variable placeholders for the options to be bound. |
| | Bind Placeholders: |
| | • ??EMIP_BIND_RESULTS_CURSOR?? |
| | For use with PL/SQL statement to bind a return cursor containing results for display. |
| | • ??EMIP_BIND_TARGET_GUID?? |
| | For use with SQL or PL/SQL to bind a target GUID. |
| | • ??EMIP_BIND_START_DATE?? |
| | For use with SQL or PL/SQL to bind a start date. |
| | • ??EMIP_BIND_END_DATE?? |
| | For use with SQL or PL/SQL to bind an end date. |
| | • ??EMIP_BIND_LOCALE_COUNTRY?? |
| | For use with SQL or PL/SQL to bind a locale country. |
| | • ??EMIP_BIND_LOCALE_LANGUAGE?? |
| | For use with SQL or PL/SQL to bind a locale language. |
| | Do not append a semi-colon (;) to the end of the SQL statement unless it is a PL/SQL statement. |

**Stacked Bar Chart**

**Table 5-53    Stacked Bar Chart**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.stacked". |
| Required | No. |
| Default Value | "false". |
| Valid Values | "true" or "false". |
| Summary | Indicates if a bar chart should be stacked. |

**Chart Title**

**Table 5-54    Chart Title**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.title". |
| Required | No. |
| Default Value | <none>. |

**Table 5-54    (Cont.) Chart Title**

| Attribute | Description |
| --- | --- |
| Summary | Chart title to identify chart for Americans with Disabilities Act compliance. |

**Width**

**Table 5-55    Width**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.width". |
| Required | No. |
| Default Value | "400". |
| Valid Values | *n*, where *n* is any String that will correctly parse to a positive integer. |
| Summary | Specifies the display width of the element in pixels. |

**Y-Axis Label**

**Table 5-56    Y-Axis Label**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.yAxisLabel". |
| Required | No. |
| Default Value | <none>. |
| Valid Values | String. |
| Summary | If this parameter is supplied, it is used as the y-axis label for charts that have an y-axis. |

**Slices as Percentage**

**Table 5-57    Slices as Percentage**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.pieShowSlicePercentLab els". |
| Required | No. |
| Default Value | <none>. |
| Valid Values | "true" or "false". |
| Summary | If this parameter is supplied, it controls whether each slice is labeled with a percentage value. This attribute is ignored for chartType attributes other than pieChart. |

**Show Values in Legend**

**Table 5-58    Show Values in Legend**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.ChartParamController.pieValuesInLegend". |
| Required | No. |
| Default Value | "value". |
| Valid Values | "percent", "value" or "none". |
| Summary | For pie charts, this parameter specifies whether values for pie slices are included in the legend along with the label for the pie slice. The default value for this attributes is "value". If specified as either "percent" or "value" then the numeric value is displayed along with the pie slice label in the form, "pie slice label (numeric value)". If "percent" is specified, then the percentage out of the total of all slice values is calculated and displayed, otherwise, the raw value of the slice is displayed. To omit a value in the legend, specify "none" as a value for this parameter. This attribute is ignored for chartType attributes other than pieChart. |

## Understanding the Metric Details Element

To declare a Metric Details element, you would use the following in the `ReportElement` tag:

```
<ReportElementParameters
element_name_nlsid=" IPMSG_METRIC_DETAILS"
element_type_ nlsid="IPMSG_ANY_TARGET_TYPE" .........>
```

**Target Type**

**Table 5-59    Target Type**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.MetDetInternalTargetType". |
| Required | No. |
| Default Value | "oracle_database". |
| Valid Values | Any valid internal target type name. |
| Summary | The type of target to be shown in the graph. |

**Metric Name**

**Table 5-60    Metric Name**

| Attribute | Description |
|---|---|
| Parameter Name | "oracle.sysman.eml.ip.render.elem.MetDetSelectedMetric". |
| Required | Yes. |
| Valid Values | Valid metric name according to target type selected. |
| Summary | Metric to be graphed. |

**Metric Column Name**

**Table 5-61    Metric Column Name**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.MetDetSelectedMetricColumn". |
| Required | Yes. |
| Valid Values | Valid column name according to the metric and target type selected. |
| Summary | Column of metric to be graphed. |

**Time Period**

**Table 5-62    Time Period**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TimePeriodParam". |
| Required | No. |
| Default Value | null. |
| Valid Values | "0:0" for last 24 Hours. |
| | "0:1" for last 7 Days. |
| | "0:2" for last 31 Days. |
| Summary | Encoded time period. |

**Width**

**Table 5-63    Width**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.MetDetWidth". |
| Required | No. |
| Default Value | 300. |
| Valid Values | $n$, where $n$ is any String that will correctly parse to a positive integer. |
| Summary | Width of the image in pixels. |

**Height**

**Table 5-64    Height**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.MetDetHeight". |
| Required | No. |
| Default Value | 300. |
| Valid Values | $n$, where $n$ is any String that will correctly parse to a positive integer. |
| Summary | Height of the image in pixels. |

**Legend Position**

**Table 5-65    Legend Position**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.MetDetLegendPosition". |
| Required | No. |
| Valid Values | "south" (default), "east". |
| Summary | Position of the legend relative to the chart. |

# Using Text Element Parameters

The Text element is used to display any message text you wish to provide for your report. To declare a Text element, include the following in the `ReportElement` tag:

```
<ReportElementParameters
element_name_nlsid="IPMSG_STYLED_TEXT"
element_type_ nlsid="IPMSG_ANY_TARGET_TYPE" ………>
```

**Message Text**

**Table 5-66    Message Text**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TextParamBean.textMessage". |
| Required | No. |
| Default Value | "" (empty String). |
| Valid Values | Any text. |
| Summary | Sets the message to display in the report. |

**Message Style**

**Table 5-67    Message Style**

| Attribute | Description |
| --- | --- |
| Parameter Name | "oracle.sysman.eml.ip.render.elem.TextParamBean.textStyleClass". |
| Required | No. |
| Default Value | "OraInstructionText". |
| Valid Values | "OraInstructionText".<br>"OraTipText". |
| Summary | Specifies the style class for the message text to adopt when displayed. |

**Link Destination**

**Table 5-68    Link Destination**

| Attribute | Description |
| --- | --- |
| Parameter Name | "ooracle.sysmn.eml.ip.render.elem.TextParamBean.textDestination". |
| Required | No. |
| Default Value | None. |
| Valid Values | Any URI. |
| Summary | Specifies an optional link destination for this text element. |

# About Report-Wide Parameters

The following parameters apply to all reporting elements within the report definition.

**Dynamic Time Selector**

You can provide a dynamic time period selector for your report definition that enables the report user to choose a specific time period with which to view the report.

If you are using Table from SQL or Chart from SQL report elements, you can structure your SQL statement such that the start and end dates will be bound automatically for you by Information Publisher. You achieve this by inserting placeholders (for example, `??EMIP_BIND_START_DATE??`) for the start and end date values as shown in the following example.

**Example: Automatic Binding of Start and End Dates**

```
'SELECT COLUMN_LABEL, ROLLUP_TIMESTAMP, AVERAGE
     FROM MGMT$METRIC_HOURLY
     WHERE TARGET_GUID = ??EMIP_BIND_TARGET_GUID??
     AND METRIC_LABEL = ''Load''
     AND KEY_VALUE = '' ''
     AND ROLLUP_TIMESTAMP > ??EMIP_BIND_START_DATE??
     AND ROLLUP_TIMESTAMP < ??EMIP_BIND_END_DATE??
     ORDER BY ROLLUP_TIMESTAMP'
```

See the online help documentation for Table from SQL or Chart from SQL for detailed information.

# Using the ImportExport.xsd File

The Information Publisher ImportExport.xsd file describes the format of the report definition XML file. The following example shows a sample NamedSql.xsd file.

**Example: Information Publisher ImportExport.xsd**

```
<xsd:schema targetNamespace="http://www.example.com/DataCenter/ReportDefinition"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:ms="http://www.example.com/DataCenter/ReportDefinition"
     elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xsd:annotation>
        <xsd:documentation>
            <strong>This is the schema definition, used by metadata services and
the report import cli. It is used to fully specify a report definiton</strong>
```

```
            </xsd:documentation>
        </xsd:annotation>


    <!-- ***************************** -->
    <!-- Main Element: ReportDefinition -->
    <!-- ***************************** -->
    <xsd:element name="ReportDefinition" type="ms:ReportDefinitionT"/>
      <!-- Defining Common Types used in a Report Definition -->
      <!-- ***************** -->
      <!-- ReportDefinitionT -->
      <!-- ***************** -->
      <!--  Documentation:
            ReportDefinitionT is type for main root element. All the Report
Definitions should validate to this type.
      -->  <xsd:complexType name="ReportDefinitionT">
       <xsd:sequence>
         <xsd:element name="ReportWideParameters" type="ms:ReportWideParametersT"
minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="ReportElement" type="ms:ReportElementT" minOccurs="0"
maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="title" type="ms:String100Def" use="required"/>
       <xsd:attribute name="description" type="ms:String500Def"/>
       <xsd:attribute name="system_report" type="ms:BooleanDef" default="0"/>
       <xsd:attribute name="category" type="ms:String100Def" use="required"/>
       <xsd:attribute name="sub_category" type="ms:String100Def" use="required"/>
       <xsd:attribute name="target_type"  type="ms:String64Def"/>
       <xsd:attribute name="is_jit_target" type="ms:BooleanDef" default="1"/>
       <xsd:attribute name="is_jit_multi_target" type="ms:BooleanDef" default="0"/>
       <xsd:attribute name="add_toc" type="ms:BooleanDef" default="0"/>
       <xsd:attribute name="pack_name" type="ms:String64Def"/>
       <xsd:attribute name="style" type="ms:String64Def" default="BLAF"/>
       <xsd:attribute name="show_navigation" type="ms:BooleanDef" default="1"/>
       <xsd:attribute name="product_name" type="ms:String100Def" default="EM"/>
       <xsd:attribute name="component_name" type="ms:String100Def"/>
       <xsd:attribute name="generate_context" type="ms:BooleanDef" default="0"/>
       <xsd:attribute name="oms_version" type="ms:NameDef" use="required"/>
     </xsd:complexType>

     <xsd:complexType name="ReportElementT">
       <xsd:sequence>
         <xsd:element name="ReportElementParameters"
         type="ms:ReportElementParametersT" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="element_name_nlsid" type="ms:String256Def"
         use="required"/>
       <xsd:attribute name="element_type_nlsid" type="ms:String100Def"
         use="required"/>
       <xsd:attribute name="header_nlsid" type="ms:String100Def"/>
       <xsd:attribute name="element_order" type="xsd:integer"/>
       <xsd:attribute name="element_row" type="xsd:integer"/>
       <xsd:attribute name="suppress_render" type="ms:BooleanDef"/>
     </xsd:complexType>

     <xsd:complexType name="ReportElementParametersT">
       <xsd:attribute name="parameterName" type="ms:String100Def" use="required"/>
       <!-- parameterValue is in CDATA, but schema definition makes no
            distinction between this and string attribute. Therefore,
            don't specify any contstraints on the attribute, thereby allowing
            it to be unbouned in length.
```

```
      -->
      <xsd:attribute name="parameterValue" type="xsd:string" use="required"/>
   </xsd:complexType>

   <xsd:complexType name="ReportWideParametersT">
      <xsd:attribute name="parameterName" type="ms:String100Def" use="required"/>
      <!-- parameterValue is in CDATA, but schema definition makes no
            distinction between this and string attribute. Therefore,
            don't specify any contstraints on the attribute, thereby allowing
            it to be unbouned in length.
      -->
      <xsd:attribute name="parameterValue" type="xsd:string" use="required"/>
   </xsd:complexType>

   <xsd:simpleType name="String64Def">
     <xsd:restriction base="xsd:string">
       <xsd:minLength value="1"/>
       <xsd:maxLength value="64"/>
       <xsd:whiteSpace value="preserve"/>
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="String100Def">
     <xsd:restriction base="xsd:string">
       <xsd:minLength value="1"/>
       <xsd:maxLength value="100"/>
       <xsd:whiteSpace value="preserve"/>
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="String500Def">
     <xsd:restriction base="xsd:string">
       <xsd:minLength value="1"/>
       <xsd:maxLength value="500"/>
       <xsd:whiteSpace value="preserve"/>
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="String256Def">
     <xsd:restriction base="xsd:string">
       <xsd:minLength value="1"/>
       <xsd:maxLength value="256"/>
       <xsd:whiteSpace value="preserve"/>
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="BooleanDef">
     <xsd:restriction base="xsd:integer">
      <xsd:enumeration value="0"/>
      <xsd:enumeration value="1"/>
     </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="NameDef">
     <xsd:restriction base="xsd:string">
       <xsd:minLength value="1"/>
       <xsd:maxLength value="64"/>
     </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

**Example: NamedSQL.xsd**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ms="http://www.example.com/DataCenter/NamedSQL"
            targetNamespace="http://www.example.com/DataCenter/NamedSQL"
            elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xsd:annotation>
       <xsd:documentation>
          <strong>This is the schema definition, used by metadata services. It is used to
fully specify a list of named sql that can be used in report definitions</strong>
       </xsd:documentation>
   </xsd:annotation>


  <xsd:element name="NamedSQLStatements">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="NamedSQL" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
             <xsd:attribute name="sqlName" type="xsd:string"/>
             <xsd:attribute name="sqlValue" type="xsd:string"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

# About Enterprise Manager Command Line Interface (EM CLI) Verbs

The following EM CLI verbs are used exclusively for report definition creation and administration.

**Example: EM CLI Verbs**

```
emcli get_reports
   [-owner="<report-owner>"]
Description:
  This verb returns a list of reports owned by or viewable
  by the user logged into the cli.
Options:
   -owner       The optional argument allows listing of viewable
                 reports owned by a specific EM user.
Output:
   The output of this report will be space separated quoted
   strings for the report title and owner with each report on it
   own line.


emcli export_report
  -title="<report-title>"
  -owner="<report-owner>"
  -output_file="<file>"
Description:
  This verb exports a report definition and all its element
  definitions given its title and owner.
Options:
   -title
```

```
                    The title of the report to export.
            -owner
                The owner of the report to export. The logged-in emcli user
                must have view privliege for the report. Target names
                will not be exported. The report is uniquely defined using
                title/owner so both must be supplied.
            -output_file
                The name of the exported file.
    Examples:
        emcli export_report \
          -title="maintenance report" \
          -owner="SHIFT1_OPERATOR" \
          -output_file="$HOME/reports/maint_report.xml"

    emcli import_report
        [-force]
        -files="file1;file2;..."
    Description:
        This verb imports a report definition from a XML file using
        the title in the xml file and the currently logged-in cli user
        as the owner of the report. If the report/owner already exists,
        the operation fails for that report with an appropriate error
        message. The report will be changed to a just-in-time report with
        the target type from the exported report. In addition, schedules
        and access privileges will need to be edited using the UI. The
        system enforces title/owner uniqueness, so an error will be thrown
        if there is already a report with the same title and owner.
    Options:
        -force
            If report with same title/owner exists, first delete it
            (and all jobs and saved copies)
        -files
            List of Path/file name(s) of XML file(s), which contains
            valid Report definition(s).
    Examples:
        emcli import_report \
          -files="$HOME/reports/maint_report1.xml;$HOME/reports/file2.xml"
```

# About Development Guidelines

Oracle recommends adhering to the guidelines in this section when defining a report definition file.

**The Component Name Must be Set to the Target Type**

The component name must be set to the target type for Enterprise Manager to associate specific report definitions with a particular metadata plug-in. For example,

```
<ReportDefinition component_name="oracle_orgchart"  … …
```

**When Using Chart from SQL and Table from SQL Elements**

- If your element accepts a single non-aggregate target (only), which is the case for most metadata plug-in target types, you can take advantage of automatic time zone date adjustment built into the Chart from SQL and Table from SQL elements by setting the `oracle.sysman.eml.ip.render.elem.adjustTimes` parameter on your element to `'true'`. When this parameter is set, the start and end dates bound to your SQL query will be adjusted from the report time zone to the target time

zone. Conversely, dates returned from the query will be adjusted from the target time zone to the report time zone.

- If your element accepts multiple targets or aggregate targets, you are responsible for handling time zone adjustment for your date values. You can obtain the report time zone from the `??EMIP_BIND_TIMEZONE_REGION??` bind variable. For the report viewer to understand the dates shown, dates displayed in a report must either conform to the report time zone or explicitly display the time zone associated with each date. The following examples illustrate common use cases.

**Example: Adjusting a Date Returned in your Select Statement from the Time Zone of a Given Target to the Report Time Zone**

```
SELECT mgmt_view_util.adjust_tz(tbl.date,
tgt.timezone_region,
??EMIP_BIND_TIMEZONE_REGION??)
FROM mgmt$target tgt, sometable tbl
WHERE <your where clause here>
```

**Example: Adjusting a Report Time Period Start and End Dates Used in the WHERE Clause of Your SELECT Statement from the Report Time Zone to your Targets Time Zone**

```
SELECT <your selected columns here>
FROM mgmt$target tgt, sometable tbl
WHERE
  tgt.target_guid = ??EMIP_BIND_TARGET_GUID?? and
  tbl.Mydate > MGMT_VIEW_UTIL.ADJUST_TZ(
    ??EMIP_BIND_START_DATE??,
    ??EMIP_BIND_TIMEZONE_REGION??,
    tgt.TIMEZONE_REGION)
AND
  tbl.Mydate < MGMT_VIEW_UTIL.ADJUST_TZ(
    ??EMIP_BIND_END_DATE??,
    ??EMIP_BIND_TIMEZONE_REGION??,
    tgt.TIMEZONE_REGION)
```

**ORACLE**

# 6
# Developing BI Publisher Reports

Oracle Business Intelligence Publisher (BI Publisher) is a strategic enterprise reporting product that provides the ability to create and manage highly formatted reports from a wide range of data sources. You can design the layout of your BI Publisher reports using Microsoft Word or Adobe Acrobat then create the reports from different types of data sources.

Oracle Enterprise Manager Cloud Control is integrated with BI Publisher. Oracle recommends using BI Publisher to create reports for Enterprise Manager Cloud Control 12*c* and later. BI Publisher includes the Data Model Editor, which is a graphical user interface for building data models within the BI Publisher interface, and the Layout Editor, which is a design tool that enables you to create report layouts within BI Publisher.

This chapter includes the following topics:

- Introduction to Oracle BI Publisher
- Training and Resources
- About the Report Data Source
- Developing a Report
- Using the Enterprise Manager EDK for Staging and Deploying BI Publisher Reports

## Introduction to Oracle BI Publisher

Plug-in developers:

1. Develop reports using data models and report templates.

   For more information about developing a report, see "Developing a Report".

2. Stage and deploy reports.

   For more information about staging and deploying reports, see "Using the Enterprise Manager EDK for Staging and Deploying BI Publisher Reports".

### Assumptions and Prerequisites

This chapter assumes you are familiar with the following:

- Management repository views against which you can write your own queries.
- Familiarity with BI Publisher.

## Training and Resources

Before you start to develop a BI Publisher report, you should take advantage of the training and reference resources available from Oracle:

- Getting Started with BI Publisher
- Oracle BI Publisher on oracle.com
- Oracle BI Publisher blog

  `http://blogs.oracle.com/xmlpublisher/`
- Oracle BI Publishing Consulting blog

  `http://bipconsulting.blogspot.com/`

# About the Report Data Source

The EMREPOS data source is available from the BI Publisher server configured for use with Enterprise Manager reports. The EMREPOS data source connects to the MGMT_VIEW account in the Management Repository and establishes the proper security context (VPD) for the Enterprise Manager user logged on to BI Publisher.

As a security measure, BI Publisher reports that use the EMREPOS data source have read-only access to the public MGMT$VIEW and GC$ views, and not to the underlying Enterprise Manager tables. This model also supports sharing report queries with Enterprise Manager users who might want to use the Enterprise Manager-provided reports as a basis for their own reports.

# Developing a Report

By default the reports and data models in the Enterprise Manager Cloud Control folder are read only. Develop your own reports in your local folders and then have a BI Publisher system administrator put the finished reports in a shared folder, outside of the Enterprise Manager Cloud Control folder.

To develop a BI Publisher report:

1. Develop your data model, based on SQL queries against the Management Repository.

   The following components are required to develop a BI Publisher report:

   - Data model
   - Report template
   - Sub template

   First, develop your data model, based on SQL queries against the Management repository, then design the layout of your report (the template) using Microsoft Word for Windows. Your template refers to one of the two common Oracle subtemplates: Portrait or Landscape.

   Review the following reports for examples of this:

   ```
   Enterprise Manager Cloud Control -> EM Sample Reports -> Targets of
   Specified Type
   ```

   ```
   Enterprise Manager Cloud Control -> EM Datamodels -> Targets of Specified
   Type
   ```

2. Develop and test the SQL queries for report data and input parameters.

3. Create a data model in BI Publisher for your data queries.

4. Create parameters in BI Publisher for your report parameters.

5. Create a report layout for your report.

   Start with the sample landscape or portrait layout RTF file provided with the Extensibility Development Kit (EDK).

6. Create and test your report.

   Download the report and data model using BI Publisher. The download option is located on the 'more...' link under the name of each report/data model.

7. Export the report file (.xdoz) and data models (.xdmz) from BI Publisher into local files.

# Using the Enterprise Manager EDK for Staging and Deploying BI Publisher Reports

To deploy BI Publisher reports from a plug-in to a BI Publisher web application:

1. Create a metadata file that adheres to the following schema:

   ```
   emcore/source/oracle/sysman/emSDK/ip/bipublisherreport/BIPublisherReport.xsd
   ```

   The following is an example of a report metadata file:

   ```
   <?xml version = '1.0' encoding = 'UTF-8' ?>
   <BIPublisherReports
           xmlns="http://www.example.com/DataCenter/BIPublisherReport">
           <ReportFile relativePath="emreports" fileName="tvmlrb104a.jar"/>
           <ReportFile relativePath="emreports" fileName="tvmlrb104b.jar"/>
   </BIPublisherReports>
   ```

2. Stage the BI Publisher reports, which are ZIP files with the extension .xdoz (report definition) or .xdmz (report data model, that is, SQL) into one or more JAR files. These files are referenced in the previous metadata file.

   The `$ORACLE_HOME/sysman/jlib/emreports.jar` file provides an example.

   ```
   $ unzip -l emreports.jar
   Archive:  emreports.jar
   Label: EMGC_MAIN_LINUX_110220
     Length      Date     Time     Name
   ---------  ---------- -----     ----
           0  02-20-2011 23:08    META-INF/
          71  02-20-2011 23:08    META-INF/MANIFEST.MF
           0  02-20-2011 23:08    bipublisherreports/
           0  02-20-2011 23:08    bipublisherreports/EM Datamodels/
        4776  02-20-2011 23:08    bipublisherreports/EM Datamodels/Usage Trend
   Report.xdmz
        4854  02-20-2011 23:08    bipublisherreports/EM Datamodels/Usage Summary
   Report.xdmz
        5008  02-20-2011 23:08    bipublisherreports/EM Datamodels/Charge Trend
   Report.xdmz
        7344  02-20-2011 23:08    bipublisherreports/EM Datamodels/Consolidation
   Reports.xdmz
        5043  02-20-2011 23:08    bipublisherreports/EM Datamodels/Charge
   Summary Report.xdmz
           0  02-20-2011 23:08    bipublisherreports/Chargeback/
       52291  02-20-2011 23:08    bipublisherreports/Chargeback/Charge Trend
   Report.xdoz
   ```

```
    66994  02-20-2011 23:08   bipublisherreports/Chargeback/Charge Summary
Report.xdoz
    26505  02-20-2011 23:08   bipublisherreports/Chargeback/Usage Trend
Report.xdoz
   112150  02-20-2011 23:08   bipublisherreports/Chargeback/Usage Summary
Report.xdoz
        0  02-20-2011 23:08   bipublisherreports/Consolidation Planner/
    50114  02-20-2011 23:08   bipublisherreports/Consolidation Planner/
Consolidation Reports.xdoz
---------                     -------
   335150                     16 files
```

3. Create your plug-in JAR files.

   The first directory in each JAR file must be `bipublisherreports`. All of the data models for the reports must be in the same subdirectory, `EM_Datamodels`, just under the `bipublisherreports` directory. For example:

   a. The plug-in, which contains all of the metadata files and BI Publisher report JAR files is installed dynamically (for metadata Plug-ins) using the plug-in environment.

   b. The BI Publisher report JAR files are placed in a subdirectory of the `metadata/bipublisherreport` directory and referenced in the metadata file (emreports).

   c. The BI Publisher reports for both platform and plug-ins are deployed to a BI Publisher web application (either at plug-in installation time or sometime later when BI Publisher is installed).

   Plug-in reports are deployed when BI Publisher is integrated with Enterprise Manager using the configureBIP script or sometime later using one of the following EMCLI verbs:

   - \* `emcli setup_bipublisher` (see Help for usage details)

   - \* `emcli deploy_bipublisher_reports[-force]`

     This last verb deploys the EM System reports (and optionally Extensibility Development Kit plug-in loaded reports) to a previously setup EM to BI Publisher relationship (using setup_bipublisher). It can also be used to upload a reports JAR file (located on the Oracle Management Server (OMS) filesystem). The operation will not overwrite existing BI Publisher reports in the EM Reports folder unless `-force` is used in the command.

     The following options are available:

     | Option | Description |
     | --- | --- |
     | [-pluginid] | In addition to Enterprise Manager system reports, you can use this option to deploy any subsequently loaded plug-in based BI Publisher reports. |
     | [-pluginversion] | This options limits the plug-ins to a specific version. |
     | [-reportsjarfile] | Use this option to deploy a single Enterprise Manager reports JAR file that contains one or more BI Publisher reports. This JAR file is located relative to the OMS's `$ORACLE_HOME`. directory. |

     For example, the syntax for the emct plug-in is:

     ```
     emcli deploy_bipublisher_reports -pluginid=oracle.sysman.emct -
     pluginversion=12.1.0.0.0
     ```

> **Note:**
>
> Do not use overlapping folders from different JAR files and PLATFORM JAR files. Doing this causes reports from the different JAR files to be placed in the same BI Publisher folder. If the same report name is referenced in multiple JAR files, there is no way of knowing which one will be deployed last.

*   `emcli sync`

    Use Sync to synchronize the reports deployed in the plugins to BI Publisher. For more information on using EMCLI sync, see sync in *Enterprise Manager Command Line Interface*.

# 7

# Collecting Target Configuration Data

This chapter provides information about defining configuration collection tables and integrating them into the Enterprise Configuration Management framework.

This chapter contains the following sections:

- Introduction to Collecting Target Configuration Data
- About the Configuration Definition Files
- Modeling Enterprise Configuration Management Tables
- Customizing the Inventory and Usage Region of the UI

## Introduction to Collecting Target Configuration Data

As a plug-in developer, you are responsible for the following steps with respect to incorporating configuration-related functions into Enterprise Configuration Management for each snapshot type:

1. Ensure that every bit of the configuration data that will be collected will not change from one collection to the next unless there is an explicit action by an administrator. Configuration data must only collect data that can change due to explicit administrator actions but should stay unchanged without such actions.

2. Define an Enterprise Configuration Management-specific metadata file that defines collection tables to match configuration collection metrics established as part of the target type definition.

   For more information, see Defining Configuration Collection Tables.

3. Name the snapshot type (the name must match the `CollectionItem` name specified on the Management Agent side) for a given target type.

   For more information, see Overview of Configuration Management Snapshot Metadata Elements .

4. Register the metadata with Enterprise Configuration Management during plug-in deployment. This creates tables for snapshot data, subject to any constraints imposed by the Enterprise Configuration Management framework. It also registers the data tables in Enterprise Configuration Management metadata tables.

   For more information, see Registering Metadata With the Configuration Management Framework.

5. Integrate with the standard Management Agent's collection mechanisms. Both `CollectionItem` and corresponding metrics must be defined at the Management Agent.

   For more information, see Modifications to Standard Collection Metrics and RAW Metrics.

6. Verify that the defined configuration collection data is returned to the Management Repository by the Management Agent.

For more information, see Testing the Configuration Collection Data and Troubleshooting.

## Assumptions and Prerequisites

This chapter assumes you are familiar with the following:

- Enterprise Manager concepts including Management Agents, metrics, and collection items
- Plug-in development overview, including how to package a plug-in and its XML files

# About the Configuration Definition Files

The metadata for configuration collection is defined in three metadata files packaged with the plug-in:

- Configuration metadata file

  This file defines the tables in the Management Repository that will store collected configuration data. In addition, the EDK includes an example of a configuration metadata file:

  ```
  \samples\plugins\HostSample\stage\oms\metadata\snapshotlive\demo_hostsample_e
  cmdef.xml
  ```

  For more information, see Defining Configuration Collection Tables.

- Target type metadata file

  Each configuration data item is collected as a metric that is defined with other metrics in the target type metadata file. In addition, the EDK includes an example of a target type metadata file:

  ```
  \samples\plugins\HostSample\stage\oms\metadata\targetType\demo_hostsample.xml
  ```

  For more information, see Creating the Target Type Metadata File.

- Default collection metadata file

  The frequency at which configuration data is collected for each metric is defined in the default collection metadata file. Metric Alert event conditions for each metric and the messages to display for such alerts are also defined in this file. In addition, the EDK includes an example of a default collection metadata file:

  ```
  \samples\plugins\HostSample\stage\oms\metadata\default_collection\demo_hostsa
  mple.xml
  ```

  For more information, see Creating the Default Collection File.

# Modeling Enterprise Configuration Management Tables

This section describes how to create configuration snapshot tables. Assume that an Oracle home target type definition is added to the Enterprise Manager framework.

You can define metadata and tables to store your configuration data and define metrics to collect the data at the Management Agent.

Each Enterprise Configuration Management metric defined in the target type metadata file corresponds to a table defined by the configuration metadata XML file. For information about target type metadata files, see Creating Target Metadata Files .

Tables can have parent-child relationships similar to foreign key constraints in the database. For every parent table row, there are *n* numbers of child table rows, and every child table row has exactly one parent table row. In effect, Enterprise Configuration Management allows trees of tables, where each table has at most one parent table. A configuration can consist of a set of these trees where no table is repeated.

From a database perspective, a table must include all key columns of its ancestor tables. (Internally, each Enterprise Configuration Management table includes an ECM_SNAPSHOT_ID column of type RAW(16) to identify the snapshot to which a given row belongs.)

Metadata for the collection tables must specify at a minimum:

- Table names, column names, column types, and hierarchical relationships between tables (if any).

- Key columns, that is columns that uniquely identify a row in a table. You must specify keys correctly. Enterprise Manager UI, comparison, and history use key columns extensively.

- UI display names that will in the Enterprise Manager generic UI.

Figure 7-1 shows the Oracle Home Entity Relationship Diagram (ERD) of the configuration tables.

**Figure 7-1    Oracle Home ERD with Tie-ins to the Framework**
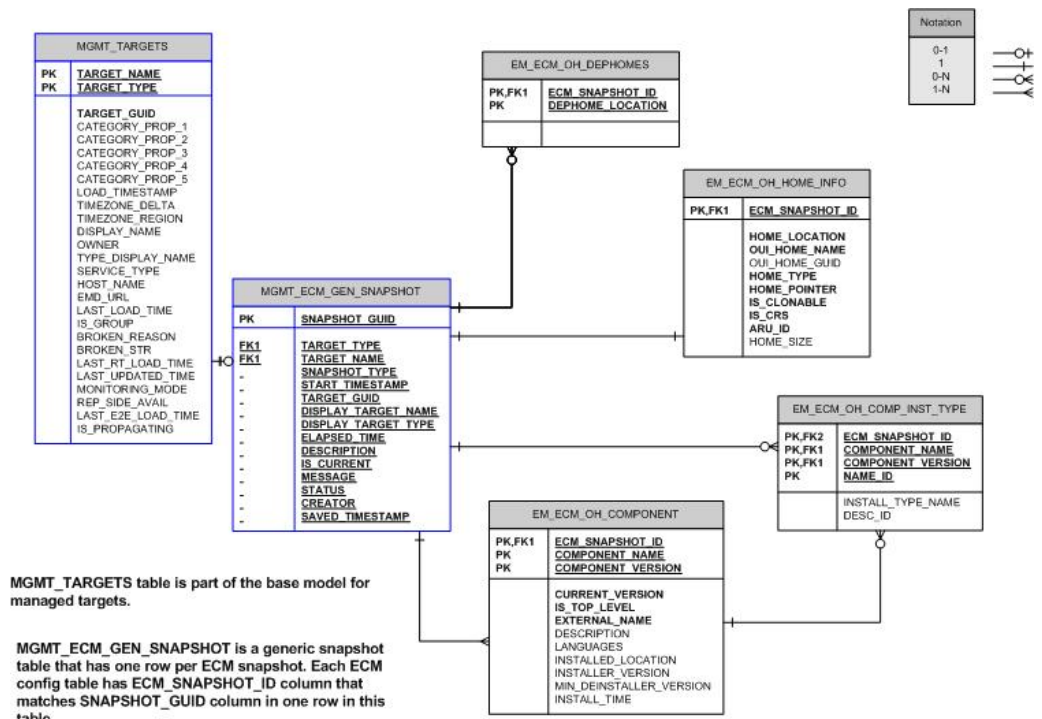
# Defining Configuration Collection Tables

The configuration metadata XML file begins with the METADATA tag. The METADATA tag establishes the relationship between the target type and the snapshot type, and defines the UI display name.

```
<METADATAS>
  <METADATA SNAP_TYPE="oracle_home_config" TARGET_TYPE="oracle_home" VER="1">

      <METADATA_UI_NAME>Oracle Home Configuration</METADATA_UI_NAME>

      CONFIGURATION COLLECTION TABLE DEFINITIONS
  </METADATA>
<METADATAS>
```

The configuration collection table definitions are encapsulated within the METADATA tag.

For more information about the elements of the configuration metadata XML, see Overview of Configuration Management Snapshot Metadata Elements . For information about the XML Schema Definitions (XSD) that governs the configuration XML files, see the Extensibility Development Kit (EDK) specifications.

## EM_ECM_OH_HOME_INFO Table

The EM_ECM_OH_HOME_INFO table stores properties related to an Oracle software installation. The metadata for this table is as follows:

```
<TABLE NAME="EM_ECM_OH_HOME_INFO" SINGLE_ROW="Y">
  <UI_NAME>Home Info</UI_NAME>
  <COLUMN NAME="HOME_LOCATION"  TYPE="STRING" TYPE_FORMAT="1024">Install
Location</COLUMN>
  <COLUMN NAME="OUI_HOME_NAME"  TYPE="STRING" TYPE_FORMAT="256" >OUI Home Name</
COLUMN>
  <COLUMN NAME="OUI_HOME_GUID"  TYPE="STRING" TYPE_FORMAT="32"  >OUI Home GUID</
COLUMN>
  <COLUMN NAME="HOME_TYPE"      TYPE="STRING" TYPE_FORMAT="1"    >Home Type</
COLUMN>
  <COLUMN NAME="HOME_POINTER"   TYPE="STRING" TYPE_FORMAT="1024">Home Pointer</
COLUMN>
  <COLUMN NAME="IS_CLONABLE"    TYPE="NUMBER" TYPE_FORMAT="1"    >Is Clonable</
COLUMN>
  <COLUMN NAME="IS_CRS"         TYPE="NUMBER" TYPE_FORMAT="1"    >Is CRS</COLUMN>
  <COLUMN NAME="ARU_ID"         TYPE="NUMBER" TYPE_FORMAT="10"   >ARU ID of the
Oracle Home</COLUMN>
  <COLUMN NAME="HOME_SIZE"      TYPE="NUMBER" TYPE_FORMAT="10"   >Size of Oracle
Home (KB)</COLUMN>
</TABLE>
```

The corresponding database table is as follows:

| COLUMN_NAME | DATA_TYPE | COLUMN_WIDTH | KEY |
| --- | --- | --- | --- |
| ECM_SNAPSHOT_ID | RAW | 16 | PK |
| HOME_LOCATION | VARCHAR2 | 1024 | |
| OUI_HOME_NAME | VARCHAR2 | 256 | |
| OUI_HOME_GUID | VARCHAR2 | 32 | |

| COLUMN_NAME | DATA_TYPE | COLUMN_WIDTH | KEY |
|---|---|---|---|
| HOME_TYPE | VARCHAR2 | 1 | |
| HOME_POINTER | VARCHAR2 | 1024 | |
| IS_CLONABLE | NUMBER | 1 | |
| IS_CRS | NUMBER | 1 | |
| ARU_ID | NUMBER | 10 | |
| HOME_SIZE (in KB) | NUMBER | 10 | |

For this table, the Primary Key consists of ECM_SNAPSHOT_ID. The `SINGLE_ROW="Y"` attribute from the metadata example indicates that each snapshot will have at most a single row in this table and therefore you do not have to mark any other columns in the table as key. This implies that the ECM_SNAPSHOT_ID column, which identifies the snapshot, will be the only key column in the table.

## EM_ECM_OH_DEP_HOMES Table

The EM_ECM_OH_DEP_HOMES table stores the locations of the Oracle home directories that a given Oracle home depends on. This data is used to form dependency associations between Oracle homes. The metadata for this table is as follows:

```
<TABLE NAME="EM_ECM_OH_DEP_HOMES">
  <UI_NAME>Dependee Oracle Homes</UI_NAME>
  <COLUMN NAME="DEP_HOME_LOCATION"  TYPE="STRING" TYPE_FORMAT="1024"
IS_KEY="Y">Dependee Home Location</COLUMN>
</TABLE>
```

The corresponding database table is as follows:

| COLUMN_NAME | DATA_TYPE | COLUMN_WIDTH | KEY |
|---|---|---|---|
| ECM_SNAPSHOT_ID | RAW | 16 | PK |
| DEP_HOME_LOCATION | VARCHAR2 | 1024 | PK |

The primary key for this table consists of ECM_SNAPSHOT_ID and DEP_HOME_LOCATION.

> **Note:**
>
> Key columns (in addition to ECM_SNAPSHOT_ID, which is always part of the key) are marked with IS_KEY="Y" in the metadata file.

## EM_ECM_OH_COMPONENT Table

The EM_ECM_OH_COMPONENT table stores information about Oracle software components in the Oracle home directory. The metadata for this table is as follows:

```
<TABLE NAME="EM_ECM_OH_COMPONENT">
  <UI_NAME>Components installed in Oracle Home</UI_NAME>
```

```
  <COLUMN NAME="COMPONENT_NAME"           TYPE="STRING" TYPE_FORMAT="128"
IS_KEY="Y">Component Name</COLUMN>
  <COLUMN NAME="COMPONENT_VERSION"        TYPE="STRING" TYPE_FORMAT="64"
IS_KEY="Y">Base Version of Component</COLUMN>
  <COLUMN NAME="CURRENT_VERSION"          TYPE="STRING"
TYPE_FORMAT="64"           >Current Version of the Component</COLUMN>
  <COLUMN NAME="INSTALL_TIME"
TYPE="TIMESTAMP"                           >Install Time</COLUMN>
  <COLUMN NAME="IS_TOP_LEVEL"             TYPE="NUMBER"
TYPE_FORMAT="1"           >Is it a top level Component</COLUMN>
  <COLUMN NAME="EXTERNAL_NAME"            TYPE="STRING"
TYPE_FORMAT="128"         >External name</COLUMN>
  <COLUMN NAME="DESCRIPTION"              TYPE="STRING"
TYPE_FORMAT="1024"        >Description</COLUMN>
  <COLUMN NAME="LANGUAGES"                TYPE="STRING"
TYPE_FORMAT="1024"        >Languages</COLUMN>
  <COLUMN NAME="INSTALLED_LOCATION"       TYPE="STRING"
TYPE_FORMAT="1024"        >Installed Location</COLUMN>
  <COLUMN NAME="INSTALLER_VERSION"        TYPE="STRING"
TYPE_FORMAT="64"          >Installer Version</COLUMN>
  <COLUMN NAME="MIN_DEINSTALLER_VERSION"  TYPE="STRING"
TYPE_FORMAT="64"          >Minimum Deinstaller Version</COLUMN>
```

> **✎ Note:**
>
> This metadata has no closing TABLE tag (yet) because it is a parent table to the next table (EM_ECM_OH_COMP_INST_TYPE Table), which is included as part of this TABLE tag.

The corresponding database table is as follows:

| COLUMN_NAME | DATA_TYPE | COLUMN_WIDTH | KEY |
|---|---|---|---|
| ECM_SNAPSHOT_ID | RAW | 16 | PK |
| COMPONENT_NAME | VARCHAR2 | 128 | PK |
| COMPONENT_VERSION | VARCHAR2 | 64 | PK |
| CURRENT_VERSION | VARCHAR2 | 64 | |
| INSTALL_TIME | DATE | - | |
| IS_TOP_LEVEL | NUMBER | 1 | |
| EXTERNAL_NAME | VARCHAR2 | 128 | |
| DESCRIPTION | VARCHAR2 | 1024 | |
| LANGUAGES | VARCHAR2 | 1024 | |
| INSTALLED_LOCATION | VARCHAR2 | 1024 | |
| INSTALLER_VERSION | VARCHAR2 | 64 | |
| MIN_DEINSTALLER_VERSION | VARCHAR2 | 64 | |

The primary key for this table consists of ECM_SNAPSHOT_ID, COMPONENT_NAME, and COMPONENT_VERSION.

## EM_ECM_OH_COMP_INST_TYPE Table

The EM_ECM_OH_COMP_INST_TYPE table stores the installation type of Oracle software components (Oracle Universal Installer (OUI) only). This a child table of EM_ECM_OH_COMPONENT Table. The metadata for this table is as follows:

```
<TABLE NAME="EM_ECM_OH_COMP_INST_TYPE">
    <UI_NAME>Install Types of Components</UI_NAME>
    <COLUMN NAME="NAME_ID"                    TYPE="STRING" TYPE_FORMAT="128"
IS_KEY="Y">Install Type Name ID</COLUMN>
    <COLUMN NAME="INSTALL_TYPE_NAME"          TYPE="STRING"
TYPE_FORMAT="128"           >Install Type Name</COLUMN>
    <COLUMN NAME="DESC_ID"                    TYPE="STRING"
TYPE_FORMAT="128"          >Install Type Description ID</COLUMN>
</TABLE>
</TABLE>
```

> **Note:**
>
> The extra closing `</TABLE>` tag is for the EM_ECM_OH_COMPONENT parent table.

The corresponding database table is as follows:

| COLUMN_NAME | DATA_TYPE | COLUMN_WIDTH | KEY |
|---|---|---|---|
| ECM_SNAPSHOT_ID | RAW | 16 | PK |
| COMPONENT_NAME | VARCHAR2 | 128 | PK |
| COMPONENT_VERSION | VARCHAR2 | 64 | PK |
| NAME_ID | VARCHAR2 | 128 | PK |
| INSTALL_TYPE_NAME | VARCHAR2 | 128 | |
| DESC_ID | VARCHAR2 | 128 | |

For this table, the primary key consists of ECM_SNAPSHOT_ID, COMPONENT_NAME, COMPONENT_VERSION, and NAME_ID.

> **Note:**
>
> COMPONENT_NAME and COMPONENT_VERSION are not listed here but are inherited from the key columns of the parent table, and the values in these columns must match a row in EM_ECM_OH_COMPONENT with the same values for ECM_SNAPSHOT_ID, COMPONENT_NAME, and COMPONENT_VERSION.

## Additional Information About the Configuration Metadata

Note the following when you are creating the configuration metadata XML file:

- Each table definition must specify its name and at least one column specification:

  **Example: Table Definition**

  ```
  <TABLE NAME="..." ...>
      Optional UI name
      Column definitions
  Optional Indexes definitions: starting 12c PS1 platform release
      Optional Child Table definitions
  </TABLE>
  ```

  Table 7-1 provides the attributes of the TABLE element.

- Table names must be specific to a given snapshot type and cannot be shared by multiple snapshot types. While internal Enterprise Manager names can start with EM_, for plug-ins and addons, Oracle recommends that you start table names with the plug-in tag (third part of a plugin ID) in upper case, followed by underscore. For example, plugin "oracle.sysman.xyz" should define tables starting with XYZ_, such as XYZ_CONFIG.

- Each column definition must specify its name and type at least:

  ```
  <COLUMN NAME="..." TYPE="..." ...>...</COLUMN>
  ```

  Table 7-1 provides the attributes of the COLUMN element.

- Columns can be of type STRING, NUMBER, TIMESTAMP, or RAW. TYPE_FORMAT is optional; its meaning derives from the value of type. For a string, it is the maximum string length. For a number, it is precision and scale, as for an Oracle database (for example, TYPE_FORMAT="4, 9").

- Specify parent child relationships between tables by nesting the TABLE tags.

- Table and view names cannot exceed 25 characters.

- Table and column order is significant. The UI display replicates the order. Import and export operations preserve the order. Delete operations (on table data) occur in inverse order. Child rows are removed before parent rows.

- COLUMN tags contain the UI display name of the column.

- Tables require key columns that uniquely identify rows in the table. Oracle recommends that the total size of the key columns is not too large (4,000 should be the maximum but much smaller sizes are acceptable).

  The key columns of all ancestor tables are automatically assumed to be inherited by the child tables and are not repeated in the child table specification.

  However, a table does not require a key if it has at most one row per parent row, or per snapshot in the case of a top-level table. Tables that do not have a key must specify the SINGLE_ROW="Y" attribute, which is set to "N" by default.

For information about the key elements of the configuration metadata, see Overview of Configuration Management Snapshot Metadata Elements .

# Overview of Configuration Management Snapshot Metadata Elements

Table 7-1 describes the key elements that define configuration management.

**Table 7-1    Key Elements of a Configuration Metadata XML File**

| Element | Description |
| --- | --- |
| METADATAS | The configuration metadata XML file starts with the METADATAS tag.<br><br>```<br><METADATAS><br>        One or more snapshot specifications<br></METADATAS><br>```<br><br>The snapshot specification corresponds to a METADATA tag. |
| METADATA | The snapshot specification corresponds to a METADATA tag and includes at least one table specification. It also defines the snapshot UI display name. It includes the following attributes:<br><br>• VER: Specifies the plug-in developer-defined metadata version and must be an integer (beginning with 1). Typically, each release increments the version (if there are changes). Only the latest ECM metadata version must be registered within a release (although Management Agents can upload older compatible versions)<br><br>• SNAP_TYPE: Names the type of snapshot. Snapshot types are defined in the context of target types. The name should begin with company name followed by an underscore. For example, oracle_.<br><br>• TARGET_TYPE: Target type for which the snapshot type metadata is defined.<br><br>• UI_IGNORE: Optional. Determines whether data is displayed as part of the configuration browser. Values are N (default), which uses UI functionality or Y, which ignores UI functionality.<br><br>• COMPARE_IGNORE: Optional. Determines whether to perform comparisons on the data. Values are N (default), which uses Compare functionality or Y, which ignores Compare functionality.<br><br>• COMPARE_UI_IGNORE: Optional. Determines whether to display data in the comparison UI. Values are N (default), which uses Compare UI functionality or Y, which ignores Compare UI functionality.<br><br>• HISTORY_IGNORE: Optional. Determines whether to track history on the data. Values are N (default), which uses History functionality or Y, which ignores History functionality.<br><br>• HISTORY_UI_IGNORE: Optional. Determines whether to display data in the history UI. Values are N (default), which uses History UI functionality or Y, which ignores History UI functionality.<br><br>**Note**: If you specify a value for a *name*_IGNORE attribute, then it is specified for all tables unless overridden at a lower (TABLE or COLUMN) level. Inheritance flows from metadata to tables, parent tables to child tables, and from all tables to their columns. |

**Table 7-1    (Cont.) Key Elements of a Configuration Metadata XML File**

| Element | Description |
| --- | --- |
| TABLE | Specifies the table name and at least one column. It includes the following attributes: |

- `NAME`: Required. Identifies the table uniquely
- `SINGLE_ROW`: Values are N (default) or Y, which indicates tables that have at most one row per parent row or at most one row per snapshot in case of top-level tables. In this latter case, no key is required. All key columns (if any) are inherited from a parent table.
- `UI_IGNORE`: Optional. Determines whether data is displayed as part of the configuration browser. Values are N (default), which uses UI functionality or Y, which ignores UI functionality.
- `COMPARE_IGNORE`: Optional. Determines whether to perform comparisons on the data. Values are N (default), which uses Compare functionality or Y, which ignores Compare functionality.
- `COMPARE_UI_IGNORE`: Optional. Determines whether to display data in the comparison UI. Values are N (default), which uses Compare UI functionality or Y, which ignores Compare UI functionality.
- `HISTORY_IGNORE`: Optional. Determines whether to track history on the data. Values are N (default), which uses History functionality or Y, which ignores History functionality.
- `HISTORY_UI_IGNORE`: Optional. Determines whether to display data in the history UI. Values are N (default), which uses History UI functionality or Y, which ignores History UI functionality.

**Note**: If you specify a value for a *name*`_IGNORE` attribute, then it is specified for all columns and a child table unless overridden at a lower (TABLE or COLUMN) level.

**Table 7-1    (Cont.) Key Elements of a Configuration Metadata XML File**

| Element | Description |
| --- | --- |
| COLUMN | Each column definition must a NAME and TYPE attributes. It includes the following attributes:<br><br>• NAME: Required. Identifies the table column uniquely.<br>• TYPE: Required. Values are STRING, NUMBER, TIMESTAMP, CLOB, or RAW<br>• TYPE_FORMAT: Optional. Value depends on the value in the TYPE attribute<br>• IS_KEY: Required. Specifies whether it is part of a key that uniquely identifies a row in the table. The key columns of all ancestor tables are automatically assumed to be inherited by the child tables.<br><br>  **Note**: A table does not require a key (unless that of ancestor table keys, if any) if it has only one row per parent row, or per snapshot in the case of a top-level table. You must specify the SINGLE_ROW="Y" attribute (which is set to "N" by default) for tables that do not have a key.<br>• UI_IGNORE: Optional. Determines whether data is displayed as part of the configuration browser. Values are N (default), which uses UI functionality or Y, which ignores UI functionality.<br>• COMPARE_IGNORE: Optional. Determines whether to perform comparisons on the data. Values are N (default), which uses Compare functionality or Y, which ignores Compare functionality.<br>• COMPARE_UI_IGNORE: Optional. Determines whether to display data in the comparison UI. Values are N (default), which uses Compare UI functionality or Y, which ignores Compare UI functionality.<br>• HISTORY_IGNORE: Optional. Determines whether to track history on the data. Values are N (default), which uses History functionality or Y, which ignores History functionality.<br>• HISTORY_UI_IGNORE: Optional. Determines whether to display data in the history UI. Values are N (default), which uses History UI functionality or Y, which ignores History UI functionality. |
| INDEXES | Optional indexes are useful when a table is used in derived associations, compliance rules, reports, or other queries where performance is important in accessing table rows based on columns other than ECM_SNAPSHOT_ID.<br><br>An optional index definitions element <INDEXES> should specify at least one <INDEX> element, and each <INDEX> element should specify name and columns:<br><br><pre><INDEXES><br>    <INDEX NAME="..." COLUMNS="..." /><br>    <INDEX NAME="..." COLUMNS="..." /><br>    …<br></INDEXES></pre><br>• NAME: Required. Identifies the table column uniquely and should be unique among all index names. Oracle recommends that the name should be the same as table name followed by _IDX1, _IDX2, and so on.<br>• COLUMN: Required. Value should be a comma-separated list of column names. |

> **Note:**
>
> A predefined ECM_METADATA_ID column is always added as the last column for each index.

The following example provides a configuration metadata XML file for an oracle_home _config snapshot. All *_IGNORE attributes keep their default values (**N**) because they are not specified in the file.

**Example: Configuration Metadata XML File**

```
<METADATAS>
  <METADATA SNAP_TYPE="oracle_home_config" TARGET_TYPE="oracle_home" VER="1"

  <METADATA_UI_NAME>Oracle Home Configuration</METADATA_UI_NAME>

  <TABLE NAME="EM_ECM_OH_HOME_INFO" SINGLE_ROW="Y">
    <UI_NAME>Home Info</UI_NAME>
    <COLUMN NAME="HOME_LOCATION" TYPE="STRING" TYPE_FORMAT="1024">Install
Location</COLUMN>
    <COLUMN NAME="OUI_HOME_NAME" TYPE="STRING" TYPE_FORMAT="256">OUI Home Name</
COLUMN>
    <COLUMN NAME="OUI_HOME_GUID" TYPE="STRING" TYPE_FORMAT="32">OUI Home GUID</
COLUMN>
    <COLUMN NAME="HOME_TYPE" TYPE="STRING" TYPE_FORMAT="1">Home Type</COLUMN>
    <COLUMN NAME="HOME_POINTER" TYPE="STRING" TYPE_FORMAT="1024">Home Pointer</
COLUMN>
    <COLUMN NAME="IS_CLONABLE" TYPE="NUMBER" TYPE_FORMAT="1">Is Clonable</COLUMN>
    <COLUMN NAME="IS_CRS" TYPE="NUMBER" TYPE_FORMAT="1">Is CRS</COLUMN>
    <COLUMN NAME="ARU_ID" TYPE="NUMBER" TYPE_FORMAT="10">ARU ID of the Oracle
Home</COLUMN>
    <COLUMN NAME="HOME_SIZE" TYPE="NUMBER" TYPE_FORMAT="10">Size of Oracle Home
(KB)</COLUMN>
  </TABLE>

  <TABLE NAME="EM_ECM_OH_DEP_HOMES">
    <UI_NAME>Dependee Oracle Homes</UI_NAME>
    <COLUMN NAME="DEP_HOME_LOCATION" TYPE="STRING" TYPE_FORMAT="1024"
IS_KEY="Y">Dependee Home Location</COLUMN>
  </TABLE>

  <TABLE NAME="EM_ECM_OH_COMPONENT">
    <UI_NAME>Components installed in Oracle Home</UI_NAME>
    <COLUMN NAME="COMPONENT_NAME" TYPE="STRING" TYPE_FORMAT="128"
IS_KEY="Y">Component Name</COLUMN>
    <COLUMN NAME="COMPONENT_VERSION" TYPE="STRING" TYPE_FORMAT="64"
IS_KEY="Y">Base Version of Component</COLUMN>
    <COLUMN NAME="CURRENT_VERSION" TYPE="STRING" TYPE_FORMAT="64">Current
Version of the Component</COLUMN>
    <COLUMN NAME="INSTALL_TIME" TYPE="TIMESTAMP" >Install Time</COLUMN>
    <COLUMN NAME="IS_TOP_LEVEL" TYPE="NUMBER" TYPE_FORMAT="1">Is a top level
Component</COLUMN>
    <COLUMN NAME="EXTERNAL_NAME" TYPE="STRING" TYPE_FORMAT="128">External name</
COLUMN>
    <COLUMN NAME="DESCRIPTION" TYPE="STRING" TYPE_FORMAT="1024">Description</
COLUMN>
    <COLUMN NAME="LANGUAGES" TYPE="STRING" TYPE_FORMAT="1024" >Languages</COLUMN>
    <COLUMN NAME="INSTALLED_LOCATION" TYPE="STRING" TYPE_FORMAT="1024">Installed
```

```
Location</COLUMN>
    <COLUMN NAME="INSTALLER_VERSION" TYPE="STRING" TYPE_FORMAT="64">Installer
Version</COLUMN>
    <COLUMN NAME="MIN_DEINSTALLER_VERSION" TYPE="STRING"
TYPE_FORMAT="64">Minimum Deinstaller Version</COLUMN>

    <TABLE NAME="EM_ECM_OH_COMP_INST_TYPE">
      <UI_NAME>Install Types of Components</UI_NAME>
      <COLUMN NAME="NAME_ID" TYPE="STRING" TYPE_FORMAT="128"
IS_KEY="Y">Install Type Name ID</COLUMN>
      <COLUMN NAME="INSTALL_TYPE_NAME" TYPE="STRING" TYPE_FORMAT="128"
>Install Type Name</COLUMN>
      <COLUMN NAME="DESC_ID" TYPE="STRING" TYPE_FORMAT="128">Install Type
Description</COLUMN>
    </TABLE>

  </TABLE>

  </METADATA>
</METADATAS>
```

> **✎ Note:**
>
> Use the information in this file to define a metric in the target type metadata
> file. For more information, see Modifications to Standard Collection Metrics
> and RAW Metrics.

# Packaging Configuration Metadata

When you have completed your configuration metadata XML file, save the file in the
following directory in your plug-in staging directory:

`oms/metadata/snapshotlive`

After you save the configuration metadata file in the plug-in staging directory, it is
available for automatic registration during plug-in deployment.

For information about the plug-in staging directory or plug-in deployment, see
Validating, Packaging, and Deploying the Plug-in .

# Registering Metadata With the Configuration Management Framework

After you complete the configuration metadata XML file, you must register it with
the Configuration Management framework. Place the configuration metadata XML
file in the `oms/metadata/snapshotlive` directory in your plug-in staging directory.
Registration takes place automatically during the installation or upgrade of the plug-in
and the necessary schema objects will be created and the configuration metadata
registered with the Configuration Management framework. For more information about
the plug-in staging directory and registration, see Validating, Packaging, and Deploying
the Plug-in .

To manually run the registration service during development of XML-only plug-ins,
enter the following:

```
emctl register oms metadata
-sysman_pwd sysman password
-pluginId plugin ID
-service LiveSnapshotRegistration
-file snapshot metadata XML file
```

**Note**: Import the plug-in into the Management Repository before the command is run.

*plugin ID* is the ID of the plug-in to which the snapshot metadata target types belong. The `LiveSnapshotRegistration` is the metadata service that creates and updates configuration management schema objects and registers configuration metadata.

This command registers the snapshot configuration file (*snapshot metadata XML file*) using SYSMAN credentials in your development environment.

Configuration Management provides the following utility that can be run optionally to generate some additional files:

```
EDK_DIR/bin/empdk generate_metadata_resource -stage_dirplugin_stage
        -service LiveSnapshotRegistration
        -input_dir input directory
        [-out_dir output directory]
        [-file_extension list of file extensions/suffixes]
        [-debug debug file name]
```

where:

- *EDK_DIR*: Directory where EDK package is unzipped. (For more information, see Installing the Extensibility Development Kit (EDK)).

- *plugin_stage*: Plug-in staging directory. (For more information, see Staging the Plug-in).

- *input directory*: Directory containing the configuration metadata XML files.

- *output directory*:Directory to contain the generated resource files. If not specified, then the resource files are generated in the current directory.

- *list of file extensions/suffixes*: Comma separated list of file extensions or suffixes for the resource files. If not specified, then all supported resource files will be generated in the output directory.

- *debug file name*: File that contains debug information. If not specified, then the default log file (createplugin.log*time*) is created in the output directory and stored warning and error messages only.

For each configuration metadata XML file in the input directory, the specified resource files are generated in the output directory with the file name `ecm_metadata_xml_file_name` as the prefix and the following supported suffixes:

- `.dlf`

  `file_name`.dlf is a file for translations of snapshot type name, table name, and column name. A generated DLF file and its translated versions should be placed in the `plugin_stage`/oms/rsc/ecm directory of the plug-in.

- `_metrix.xml, _collection.xml`

  `file_name`_metric.xml and `file_name`_collection.xml can be used as starting templates for Management Agent-side integration. For more information, see Modifications to Standard Collection Metrics and RAW Metrics.

# Supporting Translation

Data Loading Format (DLF) translation files are used to support internationalization for the display strings in plug-in metadata files so that the UI can display them in the language of the end user. Usually, you provide an original (such as English) DLF file to translators who then create similar files for other locales. All such files are loaded into the Management Repository by Enterprise Manager during the installation of a plug-in.

> **✎ Note:**
>
> Generating DLF files is optional and is required only if you require translation.

To generate DLF files for translation of the snapshot type name, table, and column names, run the following command:

```
$ORACLE_HOME/emcore/pdk/partner/bin/empdk generate_metadata_resource
-stage_dir staging directory
-service LiveSnapshotRegistration
-input_dir input directory containing snapshot metadata XML file
-file_extension extension to use for generated DLF files
-out_dir output directory to which to generate the DLF files
[-debug debug output file]
```

You must place the generated DLF files and their corresponding translated versions in the following directory of the plug-in:

```
plugin_stage/oms/rsc/ecm
```

The following manual additions are required in each DLF file:

```
<table xml:lang="en" name="MGMT_MESSAGES">
<lookup-key>
  <column name="MESSAGE_ID"/>
  <column name="SUBSYSTEM"/>
  <column name="LANGUAGE_CODE"/>
  <column name="COUNTRY_CODE"/>
</lookup-key>
<columns>
  <column name="MESSAGE_ID" type="string" maxsize="256"/>
  <column name="SUBSYSTEM" type="string" maxsize="64"/>
  <column name="LANGUAGE_CODE" type="string" language="%l"/>
  <column name="COUNTRY_CODE" type="string" language="%Cs"/>
  <column name="MESSAGE" type="string" maxsize="1000" translate="yes"/>
</columns>
<dataset>
GENERATED FILE CONTENT
</dataset>
</table>
```

**ORACLE**

# Upgrading Configuration Data

> **Note:**
>
> You must increment your integer metadata version (VER attribute) whenever you release a new version of metadata to your customers. The version should be incremented in the ECM XML metadata file (VER attribute in the METADATA element) as well as the corresponding Agent collection file.
>
> For more information about the VER attribute, see Table 7-1.

When you are upgrading existing snapshot metadata, the following changes are supported:

- New tables

- New non-key columns (these should appear after previously existing columns)

- New list of indexes (replaces any prior indexes)

- Increasing length of STRING type columns

- Values of UI_IGNORE, COMPARE_IGNORE, COMPARE_UI_IGNORE, HISTORY_IGNORE, HISTORY_UI_IGNORE and UI_NAME attributes

> **Note:**
>
> - No key columns can be added or removed
>
> - No columns formats except STRING can be altered
>
> - The length of the STRING columns only can be increased
>
> - New table cannot be added as a parent for an existing table
>
> - Tables or columns cannot be removed from the existing snapshot metadata

Take the following metadata example:

**Example: Original Metadata Definition**

```
<METADATAS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <METADATA VER="1"
    SNAP_TYPE="sn_dbconfig"
    TARGET_TYPE="sn_oracle_database"
    UI_IGNORE="Y"
    HISTORY_IGNORE="N"
    COMPARE_IGNORE="Y"
    COLLECTION_GROUP="COL_GRP_0">
  <METADATA_UI_NAME>Database Configuration</METADATA_UI_NAME>
  <TABLE NAME="TABLESPACES" DATA_SOURCE="R">
    <UI_NAME>Tablespaces</UI_NAME>
    <COLUMN NAME="TABLESPACE_NAME" TYPE="STRING" TYPE_FORMAT="30"
```

```
      IS_KEY="Y">Tablespace Name</COLUMN>
    <COLUMN NAME="SIZE" TYPE="NUMBER">Size</COLUMN>
    <COLUMN NAME="STATUS" TYPE="STRING" TYPE_FORMAT="10">Status</COLUMN>
    <TABLE NAME="DATAFILES">
      <UI_NAME>Datafiles</UI_NAME>
      <COLUMN NAME="FILE_NAME" TYPE="STRING" TYPE_FORMAT="1024"
        IS_KEY="Y">File Name</COLUMN>
      <COLUMN NAME="FILE_SIZE" TYPE="NUMBER" HISTORY_IGNORE="Y">Size</COLUMN>
      <COLUMN NAME="STATUS" TYPE="STRING" TYPE_FORMAT="9">Status</COLUMN>
    </TABLE>
  </TABLE>
 </METADATA>
</METADATAS>
```

The following example provides an example of an upgrade to the metadata definition
described in the previous example. The changes are highlighted in bold text.

**Example: Upgraded Metadata Definition**

```
<METADATAS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <METADATA VER="2"
    SNAP_TYPE="sn_dbconfig"
    TARGET_TYPE="sn_oracle_database"
    UI_IGNORE="Y"
    HISTORY_IGNORE="N"
    COMPARE_IGNORE="Y"
    COLLECTION_GROUP="COL_GRP_0">
  <METADATA_UI_NAME>Database Configuration</METADATA_UI_NAME>
  <TABLE NAME="TABLESPACES"  DATA_SOURCE="R">
    <UI_NAME>Tablespaces</UI_NAME>
    <COLUMN NAME="TABLESPACE_NAME" TYPE="STRING" TYPE_FORMAT="30"
      IS_KEY="Y">Tablespace Name</COLUMN>
    <COLUMN NAME="SIZE" TYPE="NUMBER">Size</COLUMN>
    <COLUMN NAME="STATUS" TYPE="STRING" TYPE_FORMAT="10">Status</COLUMN>
    <TABLE NAME="DATAFILES">
      <UI_NAME>Datafiles</UI_NAME>
      <COLUMN NAME="FILE_NAME" TYPE="STRING" TYPE_FORMAT="1024"
        IS_KEY="Y">File Name</COLUMN>
      <COLUMN NAME="FILE_SIZE" TYPE="NUMBER" HISTORY_IGNORE="Y">Size</COLUMN>
      <COLUMN NAME="STATUS" TYPE="STRING" TYPE_FORMAT="10">Status</COLUMN>
      <COLUMN NAME="DESC" TYPE="STRING" TYPE_FORMAT="128">Status</COLUMN>
    </TABLE>
  </TABLE>
  <TABLE NAME="TABLESPACES_1"  DATA_SOURCE="R" >
    <UI_NAME>Tablespaces_1</UI_NAME>
    <COLUMN NAME="TABLESPACE_NAME" TYPE="STRING" TYPE_FORMAT="30"
      IS_KEY="Y">Tablespace Name</COLUMN>
    <COLUMN NAME="SIZE" TYPE="NUMBER">Size</COLUMN>
    <COLUMN NAME="STATUS" TYPE="STRING" TYPE_FORMAT="10">Status</COLUMN>
    <COLUMN NAME="NOTES" TYPE="STRING" TYPE_FORMAT="128">Status</COLUMN>
    <TABLE NAME="DATAFILES_1">
      <UI_NAME>Datafiles_1</UI_NAME>
      <COLUMN NAME="FILE_NAME" TYPE="STRING" TYPE_FORMAT="1024"
        IS_KEY="Y">File Name</COLUMN>
      <COLUMN NAME="FILE_SIZE" TYPE="NUMBER" HISTORY_IGNORE="Y">Size</
COLUMN>
      <COLUMN NAME="STATUS" TYPE="STRING" TYPE_FORMAT="10">Status</COLUMN>
    </TABLE>
  </TABLE>
```

```
        </METADATA>
</METADATAS>
```

Older Enterprise Manager releases allowed for the "DROP_EXISTING_DATA" attribute in the METADATA element, but this should be removed as it is no longer supported.

Note that if you must have non-backward compatible changes in your ECM metadata, you must create a new snapshot type (that will not be comparable with old snapshot type). ECM only supports backward compatible changes.

## Modifications to Standard Collection Metrics and RAW Metrics

Enterprise Configuration Management data is collected by regular metrics, collections and Management Agent mechanisms. This data is collected though regular RAW metrics with the following modifications:

- Add a `CONFIG="TRUE"` attribute to all `Metric` and `CollectionItem` tags that collect configuration snapshot information.

- Ensure that the `CollectionItem NAME` attribute is the same as the snapshot type name (for example, oracle_home_config).

- The schedule for data collections must not specify an interval that is more frequent than once per day.

> **✎ Note:**
>
> Do not include ECM_SNAPSHOT_ID as a column in any RAW metric table descriptor

When ancestor key columns are included in child tables, you can populate a hierarchical set of tables one at a time, without having to express the hierarchical relationships during the collection. You should list parent tables before corresponding child tables in the Collection Item.

The following example provides a metric definition from the target type metadata XML file for the EM_ECM_OH_HOME_INFO table defined in the configuration metadata XML file example in Overview of Configuration Management Snapshot Metadata Elements . The information highlighted in bold font is provided by the configuration metadata XML file.

> **✎ Note:**
>
> This is just an example and additional nonconfiguration-specific Management Agent attributes might be required for your situation.

For more information about the target type metadata XML file, see Creating the Target Type Metadata File.

**Example: Defining a Metric**

```
<Metric NAME="EM_ECM_OH_HOME_INFO" TYPE="RAW" CONFIG="TRUE">
    <Display>
        <Label NLSID="…">Home Info</Label>
    </Display>
    <TableDescriptor TABLE_NAME="EM_ECM_OH_HOME_INFO">
        <ColumnDescriptor NAME="HomeLocation" COLUMN_NAME="HOME_LOCATION"
TYPE="S">
            <Display>
                <Label NLSID="…">Install Location</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="OuiHomeName" COLUMN_NAME="OUI_HOME_NAME"
TYPE="S">
            <Display>
                <Label NLSID="…">OUI Home Name</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="OuiHomeGuid" COLUMN_NAME="OUI_HOME_GUID"
TYPE="S">
            <Display>
                <Label NLSID="…">OUI Home GUID</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="HomeType" COLUMN_NAME="HOME_TYPE" TYPE="S">
            <Display>
                <Label NLSID="…">Home Type</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="HomePointer" COLUMN_NAME="HOME_POINTER"
TYPE="S">
            <Display>
                <Label NLSID="…">Home Pointer</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="IsClonable" COLUMN_NAME="IS_CLONABLE"
TYPE="N">
            <Display>
                <Label NLSID="…">Is Clonable</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="IsCrs" COLUMN_NAME="IS_CRS" TYPE="N">
            <Display>
                <Label NLSID="…">Is CRS</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="AruId" COLUMN_NAME="ARU_ID" TYPE="N">
            <Display>
                <Label NLSID="…">ARU ID of the Oracle Home</Label>
            </Display>
        </ColumnDescriptor>
        <ColumnDescriptor NAME="HomeSize" COLUMN_NAME="HOME_SIZE" TYPE="N">
            <Display>
                <Label NLSID="…">Size of Oracle Home (KB)</Label>
            </Display>
        </ColumnDescriptor>
    </TableDescriptor>
    <!-- TODO : EDIT: Edit the QueryDescriptor Element Template as required
-->
    <QueryDescriptor FETCHLET_ID="" AGENT_MODE="">
```

```
            <Property NAME="" SCOPE=""></Property>
        </QueryDescriptor>
    </Metric>
```

The following example provides a definition of the metric collection from the default collection metadata file for the metric defined in the previous example. The information highlighted in bold font is provided by the configuration metadata XML file example in Overview of Configuration Management Snapshot Metadata Elements .

For more information about the default collection metadata file, see Creating the Default Collection File.

**Example: Defining Metric Collection**

```
<CollectionItem NAME="oracle_home_config_test" CONFIG="TRUE">
        <Schedule OFFSET_TYPE="INCREMENTAL">
          <!-- Configuration Collection is done at most once every 24 Hours -->
            <IntervalSchedule INTERVAL="24" TIME_UNIT="Hr" />
        </Schedule>
        <MetricColl NAME="EM_ECM_OH_HOME_INFO" />
        <MetricColl NAME="EM_ECM_OH_DEP_HOMES" />
        <MetricColl NAME="EM_ECM_OH_CRS_NODES" />
        <MetricColl NAME="EM_ECM_OH_CLONE_PROPS" />
        <MetricColl NAME="EM_ECM_OH_COMPONENT" />
        <MetricColl NAME="EM_ECM_OH_COMP_INST_TYPE" />
        <MetricColl NAME="EM_ECM_OH_COMP_DEP_RULE" />
    </CollectionItem>
```

For a plug-in, do not forget to include the metadata and default_collection XML files at both oms and agent directories within the opar file.

# Testing the Configuration Collection Data

After integrating the configuration collection tables into the configuration management framework, you can test the configuration collection by completing the following steps:

1.  Restart the Management Agent.

    ```
    AGENT_HOME/agent/bin/emctl stop agent
    AGENT_HOME/agent/bin/emctl start agent
    ```

    In the preceding command, *AGENT_HOME* represents the Management Agent home directory.

2.  From Enterprise Manager Cloud Control, select **Targets**, then select the required target.

3.  Right-click the target and select **Configuration**, then select **Last Collected** to view the most recent data collection.

4.  Check that the required collected data is visible.

# Troubleshooting

If you are having problems with your configuration collections, do the following:

1.  Check that your snapshot type is registered in the MGMT_ECM_SNAPSHOT_METADATA table:

```
select * from mgmt_ecm_snapshot_metadata
 where target_type   = your_target_type
   and snapshot_type = your_snapshot_type;
```

You should see two rows. If not, check if there are any errors during registration in the regular log files for MRS in the following directory.

`$ORACLE_HOME/cfgtoollogs/pluginca`

2. Define the corresponding metrics to verify that collections begin and data accumulates. If you are looking at the latest collection in the UI, make sure first that the UI_IGNORE flags in the metadata are not Y for the data you are checking. If the collections are not happening, then check the following:

   • Make sure that your collection item name is the same as snapshot type and CONFIG="TRUE" is specified for both the collection item and all its metrics.

     For more information, see Modifications to Standard Collection Metrics and RAW Metrics.

   • Make sure your metrics are defined as RAW metrics and table descriptor corresponds to your ECM tables.

   • Check if the collection arrives to the Management Repository but is not updated as "current".

     In the MGMT_ECM_GEN_SNAPSHOT table, check the `is_current` status for your target and snapshot type. If there are no rows, then the collection did not progress. The IS_CURRENT flag should be set to Y to indicate the latest snapshot of data. Rows with other IS_CURRENT values are possible for internal purposes. For example, if there are no rows with the Y value, then IS_CURRENT values of T and D would indicate a snapshot started loading but did not finish.

   • Check the value of META_VER in agent target type metadata and default_collection XML files.

     During development, when any new metric or collection item is added, META_VER may need to be bumped up in these files for registration of these new entries to succeed. Check the latest instructions for Enterprise Management development regarding the META_VER value.

     For example, while during your development you may need to increase the version in order to register your changes, only one increase of the version per release is required, and therefore, while merging the code, META_VER may need to be the same as before if it were already incremented for the current release.

     From a collection perspective, you have to make sure that both new target type metadata and default_collection XML files are successfully registered and that the agent is restarted with latest files. The following commands can be used to register target type metadata and default_collection XML files:

```
emctl register oms metadata -service targetType -file target type XML
filename [-core | -pluginId Plugin Id] [ sysman_pwd "sysman password"]

emctl register oms metadata -service default_collection -file default
collection XML filename [-core | -pluginId Plugin Id] [ sysman_pwd
"sysman password"]
```

Finally, for a given target instance you are testing, make sure that its META_VER matches the META_VER of the loaded snapshot type in order to see the latest collected data that is based on your latest META_VER.

- Check the Valid-Ifs defined for the target type, snapshot type, and the metric to see if the category properties of the target instance match the Valid-Ifs. If not, the target would not show corresponding data since the configuration would not be applicable to such target.

- On the OMS Repository, check the mgmt_system_error_log table and emoms_pbs.trc/log for the snapshot type.

  You can also examine the mgmt_metric_errors table using the following command:

  ```
  select * from mgmt_metric_errors
  where target_guid = '<your target guid>'
    and coll_name = '<your snapshot type>'
  ```

  At the agent, check gcagent.log and other files in agent log directory for the same string.

- If that does not help, turn on agent backup file feature (add backupUploadedFiles=true to emd.properties and restart the agent).

  Search for your snapshot type in the following location to ensure the agent is sending data to OMS and to see what it is sending:

  ```
  agentStateDir/sysman/emd/upload/upload/succbkup/
  ```

- One potential problem you may run into relates to the configuration difference feature.

  If for some reason the configuration did not load, but the agent thinks that it has, and if the configuration does not change from that point on, the agent will keep sending short "nothing-changed" files types and the loader will keep disregarding them. To clear this issue (or just eliminate it as a potential issue while debugging), clear the agent log of config by running the following comand (no spaces around comma):

  ```
  emctl clearstate agent -incrementalconfig targetName,targetType
  ```

  Then, to initiate the collection, run the following command:

  ```
  emctl control agent runCollection targetName:targetType snapshot type
  ```

  For example:

  ```
  emctl clearstate agent -incrementalconfig
  myOracleHomeTargetName,oracle_home
  emctl control agent runCollection myOracleHomeTargetName:oracle_home
  oracle_home_config
  ```

3. At Oracle Management Service and the Management Repository, check the MGMT_SYSTEM_ERROR_LOG table and the emoms_pbs.trc file for the snapshot type. Also check the MGMT_METRIC_ERRORS table as follows:


   At the Management Agent, check the gcagent.log file and other files in agent log directory for the same string.

4. If you still have problems, turn on the Management Agent backup file feature:

   a. Open the emd.properties file.

**b.** Add the following line to the file:

```
backupUploadedFiles=true
```

**c.** Restart the Management Agent.

```
AGENT_HOME/agent/bin/emctl stop agent
AGENT_HOME/agent/bin/emctl start agent
```

In the preceding command, *AGENT_HOME* represents the Management Agent home directory.

**d.** Search for your snapshot type in the following directory to ensure that the Management Agent is sending data to the OMS:

```
agentStateDir/sysman/emd/upload/upload/succbkup/
```

**5.** A potential issue can arise relating to the configuration difference feature. If the configuration did not load but the Management Agent interprets that the configuration did load, (and if the configuration does not change), then the Management Agent sends short files indicating that nothing changed and the loader will continue to disregard the files.

To clear or eliminate this potential issue, clear the Management Agent log as follows:

```
emctl clearstate agent -incrementalconfig targetName,targetType
```

For example:

```
emctl clearstate agent -incrementalconfig myOracleHomeTargetName,oracle_home
```

Then, to initiate the collection, run the following command:

```
emctl control agent runCollection targetName:targetType snapshot_type
```

For example:

```
emctl control agent runCollection myOracleHomeTargetName:oracle_home
oracle_home_config
```

**6.** From the Cloud Control console, test the history and comparison features to see how the results look or if any flags should be tweaked.

**a.** From Enterprise Manager Cloud Control, select **Targets**, then select the required target.

**b.** Right-click the target and select **Configuration**, then select **History** to view the configuration history or select **Compare** to test the comparison feature.

> ✎ **Note:**
>
> For more information about these pages, see the Cloud Control online help.

# Customizing the Inventory and Usage Region of the UI

After configuration data is collected, Enterprise Manager provides a generic UI application that enables end users to review the configuration inventory summary.

In addition to this generic application, Oracle provides an XML interface to plug-in developers, which allows you to add your own **View By** choice to the generic Inventory and Usage region.

To add the **View By** choice:

1. Define an inventory choice metadata XML using the InvSummary.xsd file and save the file in *plugin_stage*/oms/metadata/invSummary, where *plugin_stage* is the staging directory of the plug-in. For more information about the staging directory, see Validating, Packaging, and Deploying the Plug-in . See Sample Inventory Choice XML Metadata File for an example of an inventory choice metadata XML file.

   > **Note:**
   >
   > To view the InvSummary.xsd file, from your downloaded EDK ZIP archive, go to the lib directory and locate emCoreSDK.jar. From within emCoreSDK.jar, find oracle/sysman/emSDK/core/config/ecm/inv/InvSummary.xsd.
   >
   > For more information about your downloaded EDK ZIP archive, see Installing the Extensibility Development Kit (EDK).

2. Register the inventory choice metadata XML with the UI framework. Registration takes place automatically during the installation or upgrade of the plug-in but you might want to update your XML during plug-in development without having to deploy the plug-in.

   For example, to manually run the registration service during development of XML-only plug-ins, enter the following:

   ```
   emctl register oms metadata -[pluginID pluginid]-service invSummary -file
   filename
   ```

   In the previous example, *pluginid* represents the unique identifier assigned to the plug-in and *filename* represents the full path and file name of your inventory choice metadata XML, such as /*plugin_stage*/oms/metadata/invSummary/HostsInvSummary_metadata.xml.

   For more information about MRS, see Updating Deployed Metadata Files Using the Metadata Registration Service (MRS)

## About the Inventory Choice XML

Sample Inventory Choice XML Metadata File provides an example of an Inventory Choice XML.

The following sections describe some of the sections of the XML file.

## About the InventoryChoice Element

When creating an Inventory Choice XML, include the root element, similar to the following:

```
<InventoryChoice iname="HOSTS" display_name="Hosts">
```

**Table 7-2 InventoryChoice Element**

| Element | Description |
|---|---|
| InventoryChoice | This is the root element of the XML. It defines the customization. It includes the following attributes: <br><br> • `iname`: Internal Name (NLSId in the DLF file with translated strings) <br> • `display_name`: If there is no translated string in the DLF file for `iname`, then the display name will appear. |

## About Supported Parameter Types

The following parameters are supported for this section (for a complete list of supported parameter types, refer to the XSD documentation):

• Query id: Used in the ShowByChoice section as a Bind Id.

• SHOW_BY: Rollup Type

• TARGET_NAME: Target name of the target context

• TARGET_TYPE: Target type of the target context

• MEMBER_TARGET_TYPE: Member target type filter of group home page

Parameter Type represents the type of predefined parameters mentioned in XSD.

## Applicable Target Types (Mandatory)

This section provides the list of target types for which the inventory choice is applicable.

```
<Applicable_Targettypes>
        <!--Internal target type names -->
    <!--If it has to be shown for enterprise context, give it as "enterprise"-->
            <TargetType>enterprise</TargetType>
            <TargetType>composite</TargetType>
              <TargetType>generic_system</TargetType>
              <TargetType>all</TargetType>

</Applicable_Targettypes>
```

This representation indicates that the inventory choice should be shown if region is integrated in the console home page, or group home page, or generic system home pages. If TargetType is set to **all**, then it appears by default in any inventory region on any page.

## MasterData (Mandatory)

This section provides the data source for populating the region and details page Master table.

For each inventory choice, you can have many rollup SHOWBY choices. Usually, the same data source can be used for more than one ShowBy choice because the group by clause will change only. It's better to keep the SQl queries outside the ShowBy choice sections and then reuse them inside the ShowBy choice section using the bind ID. This helps to avoid repeating the SQL query in all the ShowBy choice sections.

You can use an SQL query as a data source. As part of the Master Data section, provide the query for enterprise context and target context using a With clause. The SQL returns any columns required for ShowBy choices.

```
<Query id="HOSTS_INV_MASTER_QUERY">
<Enterprise_Ctx>

<Sql>

                                    With InvQuery AS (SELECT name ,
base_version , vendor_name ,
                        count(*) as num_hosts,…..
                            With InvQuery AS (SELECT name ,
base_version , vendor_name ,
                        count(*) as num_hosts,…..
            %GROUP_BY_SQL%
            </Sql>
    </Enterprise_Ctx>
    <!-- target context query -->
    <Target_Ctx>
          <Sql>
          ……
        </Sql>
<In_Parameters>
<Param position="1" type="TARGET_NAME"/>
…
</In_Parameters>
</Target_Ctx>
</Query>
```

%GROUP_BY_SQL% will be replaced by the SQL given in the ShowByChoice sections.

## Details data (Mandatory)

This section provides the data source for the details page from the Details table that shows data for selected master table rows. You can have a list of SQLs in this section, which can be reused in the ShowBy choice sections.

The data source for the details table should be an SQL query using Management Repository views. You can't use database tables. For information about Management Repository views, see *Oracle Enterprise Manager Cloud Control Management Repository Views Reference*.

The format is the same as the Master Table section.

For the details query, use replaceable strings %FILTER_FOR_MASTER_ROWS%, which can be replaced by selected master key row columns by the framework.

For example:

```
<Query id="HOSTS_DETAILS">
    <Enterprise_Ctx>
……
%FILTER_SELECT_FOR_MASTERROWS%
</Enterprise_Ctx>
</Query>
```

## List of Rollup Types/ShowBy Choices (Optional)

This section represents the list of rollup types.

Hosts roll up by different types such as Platform, Version, or Vendor.

For example, this is the representation for rolling up Platform.

```
<ShowByList>
    <ShowBy iname="Platform" display_name="Platform" default_selection="true">
```

In this example,

- *iname* represents the ShowBy Name mapping to the NLSID as defined in the DLF file.

- *display_name* represents the UI display name if no data is found for *iname* in the DLF file.

- *default_selection*. If this option is set to true, then it represents the default selected value in the Show By drop down list.

```
<MasterTableData>
  <SqlQuery bindid="HOSTS_INV_MASTER_QUERY">
        <!-- Group by sql will replace %GROUP_BY_SQL% given in the main
query -->
                <Group_By_Sql>
                        select name , patched, sum(num_hosts) as num_hosts
from InvQuery %FILTERDRILLDOWNVALUES% group by name,patched order by name
                </Group_By_Sql>
                            </SqlQuery>
 </MasterTableData>
```

In this example, %FILTERDRILLDOWNVALUES% is a place holder to dynamically add slice and dice dimensions of different Show By choices shown as a breadcrumb while drilling down to many levels by clicking the count bars in details page.

For example, if platform and version are the Show By choices defined, then the user would drill down to view the versions of hosts by platform , such as Windows. %FILTERDRILLDOWNVALUES% will be replaced by platform = 'Windows' dynamically in the place holder of the group by SQL.

## Target Context Query

The Target_Ctx query filters the data in the target context. For example, if the inventory choice is applicable for the group home page, then the TARGET_NAME will be filled with group target name and the TARGET_TYPE will be filled with group target type at run time when it's shown in the group target home page.Within the target_ctx SQL query, use these `<In_Parameters>` at the appropriate places to filter inside a group context page.

## UIColumnMapping Tag

The UiColumnMapping tag maps the UI column name with the backend SQL column names. It also tells if the column is visible or not.

```
<UiColumnMapping id="name" backend_column="name" isKey="true" visible="true"
uiColumn_nls_id="Platform"/>
.....
```

Date NLS format:

If the type option is specified as "date", then see the following example:

```
<UiColumnMapping id="collection_time" backend_column="collection_time"
isKey="false" visible="true" type="date"
uiColumn_nls_id="hosts_collection_time"/>
```

The UI framework will format as per the National Language Support (NLS) locale.

## DLF Files

DLF files provide the NLS source of the column names

For example:

```
<row>
    <col name="MESSAGE_ID">HOSTS</col>
    <col name="SUBSYSTEM">ECM_INV_METADATA</col>
    <col name="MESSAGE">Hosts</col>
  </row>
```

Note that the subsystem is ECM_INV_METADATA.

# Sample Inventory Choice XML Metadata File

The following example provides a sample Inventory Choice XML metadata file.

**Example: Inventory Choice XML**

```
<?xml version="1.0" encoding="UTF-8" ?>
<InventoryChoice iname="SampleHostsInv" display_name="SampleHostsInv">

 <Applicable_Targettypes>
    <TargetType>all</TargetType>
    <TargetType>enterprise</TargetType>
 </Applicable_Targettypes>
 <MasterData>
    <Query id="HOSTS_INV_MASTER_QUERY">

                <Enterprise_Ctx>

                    <Sql>
                            With InvQuery AS (
                select
                            name,
                            base_version,
                            vendor_name,
                            count(*) as num_hosts ,
                            decode(sum(num_patches) , 0 , ''No'',''Yes'') as
patched
   from
                        (
                        select os.name, os.name||'' ''||os.base_version as
base_version, os.vendor_name,host,
                            ( select
```

```
                                count(*)
                               from
                                 MGMT$OS_PATCH_SUMMARY patch
                               where
                                 patch.host =os.host and
                                 patch.target_guid = os.target_guid
                             ) as num_patches
                           from
                             MGMT$OS_SUMMARY os
                         )m

                         group by name,base_version,vendor_name
                         )

                %GROUP_BY_SQL%
                </Sql>
                </Enterprise_Ctx>
                <!-- target context query -->
                <Target_Ctx>
                    <Sql>
                With InvQuery AS (
 select
           name,
           base_version,
           vendor_name,
           count(*) as num_hosts ,
           decode(sum(num_patches) , 0 , ''No'',''Yes'') as patched
         from
         (
           select os.name, os.name||'' ''||os.base_version as base_version,
os.vendor_name,host,
               ( select
                   count(*)
                 from
                   MGMT$OS_PATCH_SUMMARY patch
                 where
                   patch.host =os.host and
                   patch.target_guid = os.target_guid
               ) as num_patches
           from
             MGMT$OS_SUMMARY os
           where os.snapshot_guid in
           (

             SELECT /*+ ORDERED */ ps.snapshot_guid
             FROM
             (
               select
                 unique t.host_name as hname
               FROM
                 mgmt$targets t,
               (
                SELECT m.assoc_target_guid as mguid
                       FROM
                         mgmt$target_flat_members  m,
                         .........
                       WHERE

                         mem.AGGREGATE_TARGET_NAME = ? and
                         mem.aggregate_target_type= ? and
                         mem.member_target_type like ? and
```

```
                        ............................
                          ) lt1
                          WHERE
                             lt1.mguid = t.target_guid
                     )lt,
                 .................
                       )
                     )
                   group by name, base_version, vendor_name
                     )
              %GROUP_BY_SQL%

                   </Sql>
                      <In_Parameters>
                      <!--Aggregate target name in which the inv choice data to be
shown -->
                          <Param position="1" type="TARGET_NAME"/>
<!-- Aggregate target type -->
                          <Param position="2" type="TARGET_TYPE"/>
                          <Param position="3" type="MEMBER_TARGET_TYPE"/>
                           ........

                      </In_Parameters>
                  </Target_Ctx>
              </Query>

  </MasterData>

  <DetailsData>
    <Query id="HOSTS_DETAILS">

                    <Enterprise_Ctx>

                         <Sql>
SELECT
         hostname,
         hwname,
         name,
         base_version,
         update_level,
         address_length_in_bits,
         vendor_name,
         freq,
         mem,
         disk,
         cpu_count,
         distributor_version,
         physical_cpu_count,
         logical_cpu_count,
         last_collection_timestamp
       FROM
       (

       SELECT
         hostname,
         hwname,
         name,
         base_version,
         update_level,
         address_length_in_bits,
```

```
                           vendor_name,
                           freq,
                           mem,
                           disk,
                           cpu_count,
                           distributor_version,
                           physical_cpu_count,
                           logical_cpu_count,
                           last_collection_timestamp
                       FROM
                       (
                           SELECT
                             o.target_name as hostname,
                             system_config  || '' '' || MA as hwname,
                             o.name as name,
                             o.name||'' ''||o.base_version as base_version,
                             o.update_level,
                             o.address_length_in_bits,
                             o.vendor_name,
                             hw.FREQ as freq,
                             hw.MEM as mem,
                             hw.DISK as disk,
                             hw.CPU_COUNT ,
                             o.DISTRIBUTOR_VERSION ,
                             hw.physical_cpu_count,
                             hw.logical_cpu_count,
                             o.LAST_COLLECTION_TIMESTAMP
                           FROM
                               MGMT$OS_HW_SUMMARY  hw  ,  MGMT$OS_SUMMARY o
                           WHERE
                             hw.host_name = o.host and
                             hw.target_guid = o.target_guid and
                             hw.SNAPSHOT_GUID = o.SNAPSHOT_GUID
                       )
                       %FILTER_SELECT_FOR_MASTERROWS% )
                       %FILTERDRILLDOWNVALUES%
                                  </Sql>
                                  </Enterprise_Ctx>
                                  <!-- target context query -->
                                  <Target_Ctx>
                                      <Sql>
                              SELECT
                       hostname,
                       hwname,
                       name,
                       base_version,
                       update_level,
                       address_length_in_bits,
                       vendor_name,
                       freq,
                       mem,
                       disk,
                       cpu_count,
                       distributor_version,
                       physical_cpu_count,
                       logical_cpu_count,
                       last_collection_timestamp
                       FROM
                       (
                                  SELECT UNIQUE
                       hostname,
```

```
            hwname,
            name,
            base_version,
            update_level,
            address_length_in_bits,
            vendor_name,
            freq,
            mem,
            disk,
            cpu_count,
            distributor_version,
            physical_cpu_count,
            logical_cpu_count,
            last_collection_timestamp
            FROM
            (
               .....--provide the target context query here for filtering in
composite target..
               ...


            ) hw_list  %FILTER_SELECT_FOR_MASTERROWS% order by hostname)
            %FILTERDRILLDOWNVALUES%

                        </Sql>
                          <In_Parameters>
                  <Param position="1" type="TARGET_NAME"/>
                  <Param position="2" type="TARGET_TYPE"/>
                  <Param position="3" type="MEMBER_TARGET_TYPE"/>
....................
                        </In_Parameters>
                      </Target_Ctx>
                  </Query>



  </DetailsData>



  <ShowByList>
    <ShowBy iname="Platform" display_name="PLATFORM" default_selection="true">
      <MasterTableData>
        <SqlQuery bindid="HOSTS_INV_MASTER_QUERY">

    <!-- Group by sql will replace %GROUP_BY_SQL% given in the main query -->
        <Group_By_Sql>
                    select name , patched, sum(num_hosts) as num_hosts from
InvQuery %FILTERDRILLDOWNVALUES% group by name,patched order by name
            </Group_By_Sql>

                </SqlQuery>
        <UiColumnMapping id="Platform" backend_column="name" isKey="true"
visible="true" uiColumn_nls_id="Platform"/>

        <UiColumnMapping id="num_hosts" backend_column="num_hosts" isKey="false"
uiColumn_nls_id="Hosts" countColumn="true"/>
        <UiColumnMapping id="patched" backend_column="patched" isKey="false"
uiColumn_nls_id="Patched"/>

      </MasterTableData>
```

```
            <DetailsTableData>
              <SqlQuery bindid="HOSTS_DETAILS"/>
                    <UiColumnMapping id="Host Name" backend_column="hostname"
isKey="true" visible="true" uiColumn_nls_id="Host Name"/>
                <UiColumnMapping id="Platform" backend_column="name" isKey="true"
visible="true" uiColumn_nls_id="Platform"/>
                <UiColumnMapping id="Version" backend_column="base_version" isKey="true"
visible="true" uiColumn_nls_id="Version"/>
                <UiColumnMapping id="Vendor" backend_column="vendor_name" isKey="true"
visible="true" uiColumn_nls_id="Vendor"/>
                <UiColumnMapping id="Hardware" backend_column="hwname" isKey="true"
visible="true" uiColumn_nls_id="Hardware"/>
                <UiColumnMapping id="Update Level" backend_column="update_level"
isKey="true" visible="true" uiColumn_nls_id="Update Level"/>


            </DetailsTableData>



    </ShowBy>
    <ShowBy iname="Version" display_name="VERSION" default_selection="false">
          <MasterTableData>
            <SqlQuery bindid="HOSTS_INV_MASTER_QUERY">

            <!-- Group by sql will replace %GROUP_BY_SQL% given in the main
query -->
                <Group_By_Sql>
                        select base_version , patched, sum(num_hosts) as
num_hosts from InvQuery  %FILTERDRILLDOWNVALUES% group by base_version,patched
order by base_version
                    </Group_By_Sql>

                </SqlQuery>

            <UiColumnMapping id="Version" backend_column="base_version"
isKey="true" visible="true" uiColumn_nls_id="Version"/>
            <UiColumnMapping id="num_hosts" backend_column="num_hosts"
isKey="false" uiColumn_nls_id="Hosts" countColumn="true"/>
            <UiColumnMapping id="patched" backend_column="patched" isKey="false"
uiColumn_nls_id="Patched"/>
            </MasterTableData>
          <DetailsTableData>
            <SqlQuery bindid="HOSTS_DETAILS"/>
                    <UiColumnMapping id="Host Name" backend_column="hostname"
isKey="true" visible="true" uiColumn_nls_id="Host Name"/>
                  <UiColumnMapping id="Platform" backend_column="name"
isKey="true" visible="true" uiColumn_nls_id="Platform"/>
                  <UiColumnMapping id="Version" backend_column="base_version"
isKey="true" visible="true" uiColumn_nls_id="Version"/>
                  <UiColumnMapping id="Vendor" backend_column="vendor_name"
isKey="true" visible="true" uiColumn_nls_id="Vendor"/>
                  <UiColumnMapping id="Hardware" backend_column="hwname"
isKey="true" visible="true" uiColumn_nls_id="Hardware"/>
                  <UiColumnMapping id="Update Level" backend_column="update_level"
isKey="true" visible="true" uiColumn_nls_id="Update Level"/>

            </DetailsTableData>

    </ShowBy>
```

```
        <ShowBy iname="Vendor" display_name="VENDOR" default_selection="false">
              <MasterTableData>
                <SqlQuery bindid="HOSTS_INV_MASTER_QUERY">

                    <!-- Group by sql will replace %GROUP_BY_SQL% given in the
main query -->
                        <Group_By_Sql>
                              select vendor_name, patched, sum(num_hosts) as
num_hosts from InvQuery %FILTERDRILLDOWNVALUES%  group by vendor_name,patched
order by vendor_name
                        </Group_By_Sql>

                </SqlQuery>
                <UiColumnMapping id="Vendor" backend_column="vendor_name"
isKey="true" visible="true" uiColumn_nls_id="Vendor"/>

                <UiColumnMapping id="num_hosts" backend_column="num_hosts"
isKey="false" uiColumn_nls_id="Hosts" countColumn="true"/>
                <UiColumnMapping id="patched" backend_column="patched"
isKey="false" uiColumn_nls_id="Patched"/>

            </MasterTableData>
            <DetailsTableData>
              <SqlQuery bindid="HOSTS_DETAILS"/>
              <UiColumnMapping id="Host Name" backend_column="hostname"
isKey="true" visible="true" uiColumn_nls_id="Host Name"/>
                  <UiColumnMapping id="Platform" backend_column="name"
isKey="true" visible="true" uiColumn_nls_id="Platform"/>
                  <UiColumnMapping id="Version" backend_column="base_version"
isKey="true" visible="true" uiColumn_nls_id="Version"/>
                  <UiColumnMapping id="Vendor" backend_column="vendor_name"
isKey="true" visible="true" uiColumn_nls_id="Vendor"/>
                  <UiColumnMapping id="Hardware" backend_column="hwname"
isKey="true" visible="true" uiColumn_nls_id="Hardware"/>
                  <UiColumnMapping id="Update Level"
backend_column="update_level" isKey="true" visible="true"
uiColumn_nls_id="Update Level"/>

            </DetailsTableData>


    </ShowBy>
  </ShowByList>
</InventoryChoice>
```

# 8

# Adding Job Types

By defining new job types, you can extend the utility and flexibility of the Enterprise Manager job system. Adding new job types also enables you to enhance corrective actions. This chapter assumes that you are already familiar with the Enterprise Manager job system.

For information about the Enterprise Manager job system, refer to the *Oracle Enterprise Manager Cloud Control Administrator's Guide*.

This chapter includes the following topics:

- About Job Types
- Introducing New Job Types
- Specifying a New Job Type in XML
- Using Commands
- About Command Error Codes
- Executing Long-Running Commands at the Oracle Management Service
- Specifying Parameter Sources
- Specifying Credential Information
- Specifying Security Information
- Specifying Lock Information
- Suspending a Job or Step
- Restarting a Job
- Adding Job Types to the Job Activity and Job Library Pages
- Examples: Specifying Job Types in XML
- About Performance Issues
- Adding a Job Type to Enterprise Manager

## Introduction to Adding Job Types

As a plug-in developer, you are responsible for the following steps with regard to adding job types:

1. Defining Job Types

   You define a job type by using an XML specification that defines the steps in a job, the work (command) that each step performs, and the relationships between the steps.

   For more information, see "About Job Types".

2. Executing long-running commands

The job system enables plug-in developers to write commands that perform their work at the Management Service level.

For more information. see"Executing Long-Running Commands at the Oracle Management Service".

3. Specifying parameter sources

By default, the job system expects plug-in developers to provide values for all job parameters, either when the job is submitted or at execution time (by adding or updating parameters dynamically).

For more information, see "Specifying Parameter Sources".

4. Specifying credential information

For more information, see "Specifying Credential Information".

5. Specifying security information

For more information, see "Specifying Security Information".

6. Specifying lock information

For more information, see"Specifying Lock Information".

7. Suspending a job or step

For more information, see "Suspending a Job or Step".

8. Restarting a job

For more information, see "Restarting a Job".

# About Job Types

Enterprise Manager enables you to define jobs of different types that can be executed using the Enterprise Manager job system, thereby extending the number and complexity of the tasks you can automate.

By definition, a job type is a specific category of job that carries out a well-defined unit of work. A job type is uniquely identified by a string. For example, OSCommand may be a job type that runs a remote command. You define a job type by using an XML specification that defines the steps in a job, the work (command) that each step performs, and the relationships between the steps.

Table 8-1 shows some of the Enterprise Manager job types and functions.

**Table 8-1    Example of Job Types**

| Job Type | Purpose |
| --- | --- |
| Backup | Backs up a database. |
| Backup Management | Performs management functions such as crosschecks and deletions on selected backup copies, backup sets, or files. |
| CloneHome | Clones an Oracle home directory. |
| DBClone | Clones an Oracle Database instance. |
| DBConfig | Configures monitoring for database releases earlier than release 10*g*. |
| Export | Exports database contents or objects within an Enterprise Manager user's schemas and tables. |

**Table 8-1　(Cont.) Example of Job Types**

| Job Type | Purpose |
|---|---|
| GatherStats | Generates and modifies optimizer statistics. |
| OSCommand | Runs an operating system command or script. |
| HostComparison | Compares the configurations of multiple hosts. |
| Import | Imports the content of objects and tables. |
| Load | Loads data from a non Oracle database into an Oracle Database. |
| Move Occupant | Moves occupants of the SYSAUX tablespace to another tablespace. |
| Patch | Patches an Oracle product. |
| Recovery | Restores or recovers a database, tablespaces, data files, or archived logs. |
| RefreshFromMetalink | Allows Enterprise Manager to download patches and critical patch advisory information from My Oracle Support (`https://support.oracle.com`). |
| Reorganize | Rebuilds fragmented database indexes or tables, moves objects to a different tablespace, or optimizes the storage attributes of specified objects. |
| Multi-Task | Runs a composite job consisting of multiple tasks. |
| SQLScript | Runs a SQL or PL/SQL script using SQL*Plus. |

# Introducing New Job Types

An Enterprise Manager job consists of a set of steps and each step runs a command or script. The job type defines how the steps are assembled. For example, which steps run serially, which ones execute in parallel, step order, and dependencies. You can express a job type, the steps, and commands in XML (for more information, see "Specifying a New Job Type in XML"). The job system then constructs an execution plan from the XML specification that enables it to run the steps in the specified order.

# Specifying a New Job Type in XML

A new job type is specified in XML. The job type specification provides the following information to the job system:

- Steps that make up the job.

- Commands or scripts to run in each step.

- How steps relate to each other. For example, whether steps run in parallel or serially, or whether one step depends on another step.

- User credentials to authenticate the job (typically, the owner of the job must provide these). The job type author must also declare these credentials in the job type XML.

- How specific job parameters should be computed (optional).

- What locks, if any, a running job execution must attempt to acquire and what happens if the locks are unavailable.

- What privileges users must have to submit a job.

The XML job type specification is then added to a metadata plug-in archive. After the metadata plug-in is added to Enterprise Manager, the job system has enough information to schedule the steps of the job, as well as what to run in each step.

## Understanding Job Type Categories

A job type can have one of the following categories depending on how it performs tasks on the targets to which it is applied:

- Single-Node

  A single-node job type is a job type that runs the same set of steps in parallel on every target on which the job is run. Typically, the target list for these job types is not fixed. They can take any number of targets. The following are examples of single-node job types:

  - OSCommand

    Runs an OS command or script on all of its targets.

  - SQL

    Runs a specified SQL script on all of its targets.

- Multi-Node or Combination

  A multi-node job type is a job type that performs different, possibly inter-related tasks on multiple targets. Typically such job types operate on a fixed set of targets. For example, a Clone job that clones an application schema might require two targets, a source database and a target database.

  > **Note:**
  >
  > You can use iterative stepsets for multi-node and combination job types to repeat the same activity over multiple targets.

## Using Agent-Bound Job Types

An Agent-bound job type is one whose jobs cannot be run unless the Management Agent of one or more targets in the target list is functioning and responding. A job type that fits this category must declare itself to be Agent-bound by setting the agentBound attribute of the jobType XML tag to true.

If a job type is Agent-bound, then the job system does not schedule any executions if one or more of the Management Agents corresponding to the targets in the target list of the job execution are not responding. The job (and all its scheduled steps) is set to a special state called *Suspended/Agent down*. The job is kept in this state until the Enterprise Manager repository tier detects that the Management Agent has restarted.

At this point, the job and its steps are set to scheduled status again and the job can execute. By declaring their job types to be Agent-bound, a job-type writer can ensure that the job system will not schedule the job when it has detected that the Management Agent is down.

> **✐ Note:**
>
> Single-node job types are Agent-bound by default while multi-node job types are not.

If an Agent-bound job has multiple targets in its target list, then it is marked as Suspended even if one of the Management Agents goes down.

An example of an Agent-bound job type is the OSCommand job type, which executes an OSCommand using the Management Agent of a specified target. However, not all job types are Agent-bound. For example, a job type that executes SQL in the Management Repository is not Agent-bound.

Enterprise Manager has a heartbeat mechanism that enables the repository tier to quickly determine when a remote Management Agent goes down. After a Management Agent is marked as Down, all Agent-bound job executions that have this Management Agent in their target list are marked Suspended/Agent Down. However, there is still a possibility that the job system might try to dispatch some remote operations during the time the Management Agent went down and when the Management Repository detects the fact. In cases when the Management Agent cannot be contacted and the step executes, the step is set back to a SCHEDULED state and is retried by the job system. The series of retries continues until the heartbeat mechanism marks the node as down, at which point the job is suspended.

When a job is marked as Suspended/Agent Down, by default the job system keeps the job in that state until the Management Agent restarts. However, there is a parameter called the grace period which, if defined, can override this behavior. The grace period is the maximum amount of time (in minutes) that a job's execution is allowed to start executing within. If the job cannot start within this grace period, the job execution is skipped for that schedule.

The only way that a job execution in a Suspended/Agent Down state can resume, is for the Management Agents to come back up. You cannot use the resume_execution() APIs to resume the job.

## About Job Steps

The unit of execution in a job is called a step. A step has a command, which determines what work the step will be doing. Each command has a Java class, called a command executor, that implements the command. A command also has a set of parameters, which will be interpreted by the command executor.

The job system offers a fixed set of pre-built commands, such as:

- the remote operation command (which executes a command remotely). For more information, see Using the remoteOp Command.

- the file transfer command that transfers a file between two Management Agents. For more information, see Using the fileTransfer Command.

- a get file command that streams a log file produced on the Management Agent tier into the Management Repository. For more information, see Using the getFile Command.

Steps are grouped into sets called stepsets. Stepsets can contain steps or other stepsets and can be categorized into the following types:

**ORACLE**

- Serial Stepsets

  Serial stepsets are stepsets where the steps execute serially. Steps in a serial stepset can have dependencies on their execution. For example, a job can specify that step S2 executes only if step S1 completes successfully, or that step S3 executes only if S1 fails.

  Steps in a serial stepset can have dependencies only on other steps or stepsets within the same stepset. By default, a serial stepset is considered to complete successfully if the last step in the stepset completed successfully. It is considered to have failed if the last step in the stepset failed. You can override this behavior by using the stepsetStatus attribute as long as the step is not a dependent on another (no successOf/failureOf/abortOf attribute).

- Parallel Stepsets

  Parallel stepsets are stepsets whose steps execute in parallel (execute simultaneously). Steps in a parallel stepset cannot have dependencies. A parallel stepset is considered to have succeeded if all the parallel steps have completed successfully. It is considered to have failed if any step within it failed. By default, a parallel stepset is considered to have failed if one or more of its constituent steps failed, and no steps were aborted. You can override this behavior by using the stepsetStatus attribute.

- Iterative Stepsets

  Iterative stepsets are special stepsets that iterate over a vector parameter. The target list of a job is available using special, implicit parameters named job_target_names and job_target_types. An iterative stepset iterates over the target list or vector parameter and essentially executes the stepset *N* times; once for each value of the target list or vector parameter.

  Iterative stepsets can execute in parallel (*N* stepset instances execute at simultaneously), or serially (*N* stepset instances are scheduled serially, one after another). An iterative stepset is said to have succeeded if all its *N* instances have succeeded. Otherwise, it is said to have failed if at least one of the *N* stepsets aborted. It is said to have failed if at least one of the *N* stepsets failed and none were aborted. An abort always causes an iterative stepset to stop processing further.

  Steps within each iterative stepset instance execute serially and can have serial dependencies similar to those within serial stepsets. Iterative serial stepsets have an attribute called iterateHaltOnFailure (not applicable for iterativeParallel stepsets). If this is set to true, the stepset halts at the first failed or aborted child iteration. By default, all iterations of an iterative serial stepset execute, even if some of them fail (iterateHaltOnFailure=false).

- Switch Stepsets

  Switch stepsets are stepsets where only one of the steps in the stepset is executed based on the value of a specified job parameter. A switch stepset includes a switchVarName attribute, which is a job (scalar) parameter with a value that is examined by the job system to determine which of the steps in the stepset must be executed. Each step in a switch stepset has a switchCaseVal attribute, which is one of the possible values of the parameter specified by switchVarName.

  The step in the switch stepset that is executed is the one whose switchCaseVal parameter value matches the value of the switchVarName parameter of the switch stepset. Only the selected step in the switch stepset is executed. Steps in a switch stepset cannot have dependencies with other steps or stepsets within the same stepset or outside.

By default, a switch stepset is considered to complete successfully if the selected step in the stepset completed successfully. It is considered to have failed if the selected step in the stepset failed. Also, a switch stepset succeeds if no step in the stepset was selected.

For example, if there is a switch stepset with two steps, S1 and S2 and you specify the following:

– switchVarName is sendEmail

– switchCaseVal for S1 is true

– switchCaseVal for S2 is false

If the job is submitted with the job parameter sendEmail set to true, then S1 will be executed. If the job is submitted with the job parameter sendEmail set to false, then S2 will be executed. If the value of sendEmail is anything else, the stepset still succeeds but does nothing.

* Nested Jobs

One of the steps in a stepset might itself be a reference to another job type. A job type can include other job types within itself. However, a job type cannot reference itself.

Nested jobs are a convenient way to reuse blocks of functionality. For example, performing a database backup is a job with a complicated sequence of steps. However, other job types (such as patch and clone) might use the backup facility as a nested job. With nested jobs, the job type writer can choose to pass all the targets of the containing job to the nested job, or only a subset of the targets. Also, the job type can specify whether the containing job should pass all its parameters to the nested job or whether the nested job has its own set of parameters (derived from the parent job's parameters).

The status of the individual steps and stepsets (and possibly other nested jobs) within the nested job determines the status of a nested job.

> **Note:**
>
> If a nested job refers to a job type with singleTarget set to true, then you must explicitly specify the target type applicable for the nested job, using the targetType attribute of the nested job. Without this, the nested job picks those targets that correspond to its job type's default target type only.

## Affecting the Status of a Stepset

The default algorithm by which the status of a stepset is computed from the status of its steps can be altered by the job type, using the stepsetStatus attribute of a stepset. By setting stepsetStatus to the name (ID) of a step, stepset, or job contained within it, a stepset can indicate that the status of the stepset depends on the status of the specific step, stepset, or job named in the stepStatus attribute. This feature is useful if the author of a job type wants a stepset to succeed, even if certain steps within it fail.

An example is a step that runs as the final step in a stepset in a job that sends e-mails about the status of the job to a list of administrators. The status of the job must be set to the status of the step (or steps) that performs the work, and not the status of the

step that sent the e-mail. Only steps that are unconditionally executed can be named in the stepsetStatus attribute. A step, stepset, or job that is executed as a successOf or failureOf dependency cannot be named in the stepsetStatus attribute.

## Passing Job Parameters

To pass the parameters of the job to steps, enclose the parameter name in a placeholder (contained within two % symbols). For example, %patchNo% represents the value of a parameter named patchNo. The job system substitutes the value of this parameter when it is passed to the command executor of a step.

Placeholders can also be defined for vector parameters by using the [] notation. For example, the first value of a vector parameter called patchList is referenced as %patchList%[1], the second is %patchList%[2].

The job system provides a predefined set of placeholders that can be used. These are always prefixed by job_. The following placeholders are provided:

- job_iterate_index

    The index of the current value of the parameter in an iterative stepset, when iterating over any vector parameter. The index refers to the closest enclosing stepset only. In case of nested iterative stepsets, the outer iterate index cannot be accessed.

- job_iterate_param

    The name of the parameter being iterated over, in an iterative stepset.

- job_target_names[n]

    The job target name at position n. For single-node jobs, the array would always be only of size 1 and refer only to the current node the job is execution on, even if the job was submitted against multiple nodes.

- job_target_types[n]

    The type of the job target at position n. For single-node jobs, the array would always only be of size one and refer only to the current node the job is executing on, even if the job was submitted against multiple nodes.

- job_name

    The name of the job.

- job_type

    The type of the job.

- job_owner

    The Enterprise Manager user that submitted the job.

- job_id

    The job id. This is a string representing a globally unique identifier (GUID).

- job_execution_id

    The execution id. This is a string representing a GUID.

- job_step_id

    The step id. This is an integer.

In addition to the above placeholders, the following target-related placeholders are also supported:

- emd_root: The location of the Management Agent installation
- perlbin: The location of the (Enterprise Manager) Perl installation
- scriptsdir: The location of Management Agent-specific scripts

The above placeholders are not interpreted by the job system, but by the Management Agent. For example, when %emd_root% is used in the remoteCommand or args parameters of the `remoteOp` command, or in any of the file names in the `putFile`, `getFile` and `transferFile` commands, the Management Agent substitutes the actual value of the Management Agent root location for this placeholder.

## About Job Step Output and Errors

A step consists of a status (indicates whether it succeeded, failed, or terminated), some output (the log of the step), and an error message. If a step failed, the command executed by the step could indicate the error in the error message column. By default, the standard output and standard error of an asynchronous remote operation is set to the output of the step that requested the remote operation.

A step can choose to insert error messages by using either:

- the getErrorWriter() method in CommandManager (synchronous)
- the insert_step_error_ message API in the mgmt_jobs package (typically, this is called by a remotely executing script in a command channel)

# Using Commands

This section describes available commands and associated parameters. Targets of any type can be provided for the target names and target type parameters described in the following sections. The job system automatically identifies and contacts the Management Agent that is monitoring the specified targets.

## Using the remoteOp Command

The remote operation command has the identifier `remoteOp`. The command accepts a credential usage with name as `defaultHostCred`, which you must have to perform the operation on the host of the target. The binding can be performed as follows:

```
<step ID="Step_2" command="remoteOp">
  <credList>
    <cred usage="defaultHostCred" reference="osCreds"/>
  </credList>
  <paramList>
    <param name="targetName">%job_target_names%[1]</param>
    <param name="targetType">%job_target_types%[1]</param>
    <param name="remoteCommand">%remoteCommand%</param>
    <param name="args">%args%</param>
    <param name="executeSynchronous">false</param>
  </paramList>
</step>
```

defaultHostCred is the credential usage which is understood by the command. For example, the Java code in the command makes a call for credentials with this string, whereas osCreds is the credential usage declared in the job type at the top level.

The remote operation command takes the following parameters:

- remoteCommand: The path name to the executable/script (for example, /usr/local/bin/perl).

- args: A comma-separated list of arguments to the remoteCommand.

- targetName: The name of the target on which the command is executed. You can use placeholders to represent targets.

- targetType: The target type of the target on which the command is executed.

- executeSynchronous: This option defaults to false whereby a remote command always executes asynchronously on the Management Agent and updates the status of the step after the command is executed.

  If this option is set to true, then the command executes synchronously, waiting until the Management Agent completes the process. Typically, this parameter is set to true for quick, short-lived remote operations, such as starting up a listener. For remote operations that take a long time to execute, this parameter must be set to false.

  > **Note:**
  >
  > This parameter is set to false and you cannot override the setting.

- successStatus: A comma-separated list of integer values that determines the success of the step. If the remote command returns any of these numbers as the exit status, then the step is successful. The default is zero. These values are only applicable when executeSynchronous is set to true.

- failureStatus: A comma-separated list of integer values that determines the failure of the step. If the remote command returns any of these numbers as the exit status, the step has failed. The default is all nonzero values. These values are only applicable when executeSynchronous is set to true.

- input: If specified, this is passed as standard input to the remote program.

- outputType: Specifies the type of output the remote command generates. This option can have two values:

  - Normal (default)

    Normal output is output that is stored in the log corresponding to this step and is not interpreted in any way.

  - Command

    Command output is output that can contain one or more command blocks, which are XML sequences that map to preregistered SQL procedure calls. This option enables remote commands to generate command blocks that can be directly loaded into schema in the Management Repository.

The standard output generated by the executed command is stored by the job system as the output corresponding to this step.

## Using Auxiliary Credentials

In some cases, it might be necessary to pass on additional credentials for a remote operation. These credentials are called auxiliary credentials because they are used in addition to the host credentials required to connect to the Management Agent where the remote operation must be spawned.

The processing of auxiliary credentials depends on the process that is spawned at the Management Agent. The job system provides a mechanism to extract the column values of a credential into variables that can be used for substitution within the input parameter of the remote operation command (`remoteOp`).

To use this option, you must define the auxiliary credential usage in the job type and the credential type for which it should be appropriately set.

To consume the credential within a remote operation, use the following:

```
<step ID="Command" command="remoteOp">
  <credList>
    <cred usage="defaultHostCred" reference="defaultHostCred"/>
    <cred usage="defaultDBCred" reference="defaultDBCred">
      <map toParam="db_username" credColumn="DBUserName"/>
      <map toParam="db_password" credColumn="DBPassword"/>
      <map toParam="db_role" credColumn="DBRole"/>
    </cred>
  </credList>
  <paramList>
    <param name="remoteCommand">%perlbin%/perl</param>
    <param name="args">-I,%emd_root%/sysman/admin/scripts/
jobutil,mylocation/runSQL.pl,%OracleHome%, %SID%, %sqlplus_args%</param>
    <param name="input"><![CDATA[
        __EM_JOB_SQL_USER__=%db_username%
        __EM_JOB_SQL_PASSWORD__=%db_password%
        __EM_JOB_SQL_DBROLE__=%db_role%
        __EM_JOB_INPUT_STREAM_END__
          ]]>
    </param>
    <param name="targetName">%job_target_names%[1]</param>
    <param name="targetType">%job_target_types%[1]</param>
  </paramList>
</step>
```

## Using the fileTransfer Command

The `fileTransfer` command transfers a file from one Management Agent to another. It can also execute a command on the source Management Agent and transfer its standard output as a file to the destination Management Agent or as standard input to a command on the destination Management Agent. The `fileTransfer` command is always asynchronous and it takes the following parameters:

```
<step ID="S1" command="fileTransfer">
  <credList>
      <cred usage="srcReadCreds" reference="mySourceReadCreds"/>
      <cred usage="dstWriteCreds" reference="myDestWriteCreds"/>
  </credList>
    <paramList>
      <param name="sourceTargetName">%job_target_names%[1]</param>
      <param name="sourceTargetType">%job_target_types%[1]</param>
```

```
            <param name="destTargetName">%job_target_names%[2]</param>
            <param name="destTargetType">%job_target_types%[2]</param>
            <param name="sourceFile">%sourceFile%</param>
            <param name="sourceCommand">%sourceCommand%</param>
            <param name="sourceArgs">%sourceArgs%</param>
            <param name="sourceInput">%sourceInput%</param>
            <param name="destFile">%destFile%</param>
            <param name="destCommand">%destCommand%</param>
            <param name="destArgs">%destArgs%</param>
    </paramList>
</step>
```

The following command uses two credentials. The `srcReadCreds` credential is used to read the file from the source and the `dstWriteCreds` credential is used to write the file to the destination. The binding can be performed as follows:

```
<step ID="S1" command="fileTransfer">
      <credList>
         <cred usage="srcReadCreds" reference="mySourceReadCreds"/>
         <cred usage="dstWriteCreds" reference="myDestWriteCreds"/>
      </credList>
  <paramList>
    <param name="sourceTargetName">%job_target_names%[1]</param>
    <param name="sourceTargetType">%job_target_types%[1]</param>
    <param name="destTargetName">%job_target_names%[2]</param>
    <param name="destTargetType">%job_target_types%[2]</param>
    <param name="sourceFile">%sourceFile%</param>
    <param name="sourceCommand">%sourceCommand%</param>
    <param name="sourceArgs">%sourceArgs%</param>
    <param name="sourceInput">%sourceInput%</param>
    <param name="destFile">%destFile%</param>
    <param name="destCommand">%destCommand%</param>
    <param name="destArgs">%destArgs%</param>
  </paramList>
</step>
```

- `sourceTargetName`: The target name corresponding to the source Management Agent.

- `destTargetName`: The target name corresponding to the destination Management Agent.

- `destTargetType`: The target type corresponding to the destination Management Agent.

- `sourceFile`: The file to be transferred from the source Management Agent.

- `sourceCommand`: The command to be executed on the source Management Agent. If this is specified, then the standard output of this command is streamed to the destination Management Agent. Both `sourceFile` and `sourceCommand` parameters cannot be specified.

- `sourceArgs`: A comma-separated set of command-line parameters for the `sourceCommand`.

- `destFile:` The location or file name of where the file is to be stored on the destination Management Agent.

- `destCommand`: The command to be executed on the destination Management Agent. If this is specified, then the stream generated from the source Management Agent (whether from a file or a command) is sent to the standard input of this command. You cannot specify both `destFile` and `destCommand` parameters.

- `destArgs`: A comma-separated set of command-line parameters for the `destCommand`.

The `fileTransfer` command succeeds (and returns a status code of 0) if the file was successfully transferred between the Management Agents. If there was an error, it returns error codes appropriate to the reason for failure.

## About the putFile Command

The `putFile` command enables you to transfer large amounts of data from the Management Repository to a file on the Management Agent. The transferred data can come from a Binary Large Objects (BLOB) in the Management Repository, a file on the file system, or embedded in the specification (inline).

If a file is being transferred, the location of the file must be accessible from the Management Repository installation. If a BLOB in a Management Repository is being transferred, then it must be in a table in the Management Repository that is accessible to the Management Repository schema user (typically mgmt_rep).

The command accepts a credential usage with name as `defaultHostCred`. You must have these credentials to write the file at the host of the target. The binding can be performed as follows:

```
<step ID="S1" command="putFile">
     <credList>
         <cred usage="defaultHostCred" reference="osCreds"/>
     </credList>
   <paramList>
    <param name="sourceType">file</param>
    <param name="targetName">%job_target_names%[1]</param>
    <param name="targetType">%job_target_types%[1]</param>
    <param name="sourceFile">%oms_root%/myfile</param>
    <param name="destFile">%emd_root%/yourfle</param>
   </paramList>
</step>
```

The `putFile` command requires the following parameters:

- `sourceType`: The type of the source data. This can be SQL, file, or inline.

- `targetName`: The name of the target where the file is to be transferred (destination Management Agent).

- `targetType`: The type of the destination target.

- `sourceFile`: The file to be transferred from the Management Repository (if `sourceType` is set to `fileSystem`). This must be a file that is accessible to the Management Repository installation.

- `sqlType`: The type of SQL data (if the `sourceType` is set to sql). Valid values are CLOB and BLOB.

- `accessSql`: A SQL statement that is used to retrieve the BLOB data (if the `sourceType` is set to sql). For example, " select output from `my_output_table` where blob_id=%blobid%".

- `destFile`: The location or file name of where the file is to be stored on the destination Management Agent.

- contents: If the sourceType is set to "inline", this parameter contains the contents of the file. Note that the text can include placeholders for parameters in the form %param%.

The putFile command succeeds if the file was transferred successfully and the status code is set to 0. On failure, the status code is set to an integer indicating the reason for failure.

## Using the getFile Command

The getFile command transfers a file from a Management Agent to the Management Repository. The file is stored as the output of the step that executed this command.

The command accepts a credential usage with the name as defaultHostCred, which you must have to read the file at the host of the target. The binding can be performed as follows:

```
<step ID="S1" command="getFile">
      <credList>
        <cred usage="defaultHostCred" reference="osCreds"/>
      </credList>
   <paramList>
     <param name="targetName">%job_target_names%[1]</param>
     <param name="targetType">%job_target_types%[1]</param>
     <param name="sourceFile">%sourceFile%</param>
     <param name="destType">%destType%</param>
     <param name="destFile">%destFile%</param>
     <param name="destParam">%destParam%</param>
   </paramList>
</step>
```

The getFile command has the following parameters:

- sourceFile: The location of the file to be transferred to the Management Agent.

- targetName: The name of the target where the Management Agent will be contacted to get the file.

- targetType: The type of the target.

The getFile command succeeds if the file was transferred successfully and the status code is set to 0. On failure, the status code is set to an integer indicating the reason for failure.

## Using the execAndSuspend Command

The execAndSuspend command is similar to the remoteOp command but it is used for executing a host process that restarts the Management Agent. Typically, use this command in scenarios that update Management Agent binaries or configuration and require a restart of the Management Agent. The command "posts" the Agent-based operation to the Management Agent and switches its status to "success" immediately while the subsequent step moves into a suspended status waiting for the "startup" notification from the Management Agent.

It is important to follow these restrictions and guidelines:

- The command executed at the Management Agent must not produce any standard output or errors. Such output, if any, must be redirected to a file or to null as part of the submitted operation. Failure to do this could cause the command to fail.

- The job type must contain a step immediately after a step that runs the `execAndSuspend` command. This successor step checks the success of the operation that was submitted as part of the `execAndSuspend` step. Because the Agent-based operation might have failed, the successor step must avoid using `remoteOp` and rely on direct Agent-based Java calls to check the status of the operation.

Most of the arguments to this command are similar to the `remoteOp` command. This command accepts a credential usage with name as `defaultHostCred`, which you must have to perform the operation on the host of the target. The binding can be performed as follows:

```
<step ID="Ta_S1_suspend" command="execAndSuspend">
     <credList>
         <cred usage="defaultHostCred" reference="osCreds"/>
     </credList>
  <paramList>
    <param name="remoteCommand">%command%</param>
    <param name="args">%args%</param>
    <param name="targetName">%job_target_names%[1]</param>
    <param name="targetType">%job_target_types%[1]</param>
    <param name="suspendTimeout">2</param>
  </paramList>
</step>
```

The `execAndSuspend` command has the following parameters:

- `remoteCommand`: The path name to the executable or script, such as /usr/local/bin/perl.

- `args`: A comma-separated list of arguments to the remoteCommand

- `targetName`: The name of the target on which the command is executed. You can use placeholders to represent targets

- `targetType`: The target type of the target on which the command is executed.

- `input`: If specified, this is passed as standard input to the remote program.

- `suspendTimeout`: The duration, in minutes, to wait for the notification of the Management Agent's startup. If the notification is not received within this time, the execution resumes and the successor step is executed. (The successor step is also executed if the Management Agent's startup notification is received, so the successor step must determine whether it timed out or completed successfully).

Here `defaultHostCred` is the credential usage which is understood by the command. For example, the Java code in the command would make a call for credential with this string, whereas the `osCreds` is the credential usage declared in the job type at the top level.

# About Command Error Codes

The `remoteOp`, `putFile`, `fileTransfer` and `getFile` commands return the error codes listed in Table 8-2. In the following messages, "command process" refers to a process that the Management Agent executes that actually executes the specified remote command and grabs the standard output and standard error of the executed command.

On a UNIX installation, this process is called nmo and is located in $EMD_ROOT/bin. It must be SETUID to root before it can be used successfully. This does not pose

a security risk because nmo will not execute any command unless it has a valid username and password.

**Table 8-2    Command Error Codes**

| Error Code | Description |
| --- | --- |
| 0 | No error. |
| 1 | Could not initialize core module. Most likely, something is wrong with the installation or environment of the Agent. |
| 2 | The Agent ran out of memory. |
| 3 | The Agent could not read information from its input stream. |
| 4 | The size of the input parameters was too large for the Agent to handle. |
| 5 | The command process was not setuid to root. (Every UNIX Agent installation has an executable called nmo, which must be setuid root). |
| 6 | The specified user does not exist on this system. |
| 7 | The password was incorrect. |
| 8 | Could not run as the specified user. |
| 9 | Failed to fork the command process (nmo). |
| 10 | Failed to execute the specified process. |
| 11 | Could not obtain the exit status of the launched process. |
| 12 | The command process was interrupted before exit. |
| 13 | Failed to redirect the standard error stream to standard output. |

# Executing Long-Running Commands at the Oracle Management Service

The job system enables plug-in developers to write commands that perform their work at the Management Service level. For example, a command that reads two Large Objects (LOBs) from the database and performs various transformations on them and writes them back. The job system expects such commands to implement an (empty) interface called LongRunningCommand, which is an indication that the command executes synchronously on the middle tier, and could potentially execute for a long time. This enables a component of the job system called the dispatcher to schedule the long-running command as efficiently as possible, so as not to degrade the throughput of the system.

## Configuring the Job Dispatcher to Handle Long-Running Commands

The dispatcher is a component of the job system that executes the various steps of a job when they are ready to execute. The command class associated with each step is called and any asynchronous operations requested by it are dispatched; a process referred to as dispatching a step. The dispatcher uses thread-pools to execute steps. A thread-pool is a collection of a specified number of worker threads, any one of which can dispatch a step.

The job system dispatcher uses two thread-pools:

- a short-command pool for dispatching asynchronous steps and short synchronous steps

- a long-command pool for dispatching steps that have long-running commands

Typically, the short-command pool has a larger number of threads (for example, 25) compared to the long-running pool (for example, 10).

Usually the long-running middle-tier steps are few compared to more numerous, short-running commands. However, the sizes of the two pools are fully configurable in the dispatcher to suit the job mix at a particular site. Because multiple dispatchers can run on different nodes, the site administrator can dedicate a dispatcher to only dispatch long-running or short-running steps.

## Specifying Parameter Sources

By default, the job system expects plug-in developers to provide values for all job parameters, either when the job is submitted or at execution time (by adding or updating parameters dynamically). Typically, an application supplies these parameters in one of the following ways:

- Asking the user of the application at the time of submitting the job.

- Fetching parameter values from application-specific data (such as a table) and then inserting them into the job parameter list.

- Generating new parameters dynamically through the command blocks in the output of a remote command. These could be used by subsequent steps.

The job system offers the concept of parameter sources so that plug-in developers can simplify the amount of application-specific code they have to write to fetch and populate job or step parameters (such as the second category above). A parameter source is a mechanism that the job system uses to fetch a set of parameters, either when a job is submitted or when it is about to start executing.

The job system supports SQL (a PL/SQL procedure to fetch a set of parameters), credential (retrieval of username and password information from the Enterprise Manager credentials table) and user sources. Plug-in developers can use these pre-built sources to fetch a wide variety of parameters. When the job system has been configured to fetch one or more parameters using a parameter source, you do not have to specify the parameters in the parameter list to the job when a job is submitted. The job system automatically fetches the parameters and adds them to the parameter list of the job.

A job type can embed information about the parameters that must be fetched by having an optional paramInfo section in the XML specification. The following example provides a snippet of a job type that executes a SQL query on an application-specific table to fetch three parameters, a, b, and c.

```
<jobType version="1.0" name="OSCommand" >
<paramInfo>
    <!-- Set of scalar params -->
    <paramSource paramNames="a,b,c" sourceType="sql" overrideUser="true">
        select name, value from name_value_pair_table where
            name in ('a', 'b', 'c');
    </paramSource>
</paramInfo>
.... description of job type follows ....
</jobType>
```

In the previous example, the paramInfo section contains the following elements:

- `paramSource`: Each paramSource tag references a parameter source that can be used to fetch one or more parameters.

- `paramNames`: The paramNames attribute is a comma-separated set of parameter names that the parameter source is expected to fetch.

- `sourceType`: The sourceType attribute indicates the source that will be used to fetch the parameters (one of sql, credential or user)

- `overrideUser`: The overrideUser attribute, if set to true, indicates that this parameter-fetching mechanism will always be used to fetch the value of the parameters, even if the parameter was specified by the user (or application) at the time the job was submitted. The default for the overrideUser attribute is false, indicating that the parameter source mechanism will be disabled if the parameter was already specified when the job was submitted.

  You can add additional source-specific properties to a parameter source that describes the fetching mechanism in greater detail. Understanding SQLParameter Source provides more information.

- `evaluateOnRetry`: The evaluateOnRetry attribute is an optional attribute, applicable for all. The default setting is false for all, except credentials (credentials ignores the value set and forces true). It indicates whether the parameter source must be run again when a failed execution of this job type is retried.

## Understanding SQLParameter Source

The SQL parameter source enables plug-in developers to specify a SQL query or a PL/SQL procedure that fetches a set of parameters.

## Using a PL/SQL Procedure to Fetch Scalar and Vector Parameters

The job type XML syntax is as follows:

```
<paramSource sourceType="sql" paramNames="param1, param2, ...">
  <sourceParam name="procName"   value="MyPackage.MyPLSQLProc"/>
  <sourceParam name="procParams" value="%a%, %b%[1], ..."/>
</paramSource>
```

The values specified in `paramNames` are the names of the parameters that are expected to be returned by the PL/SQL procedure specified in `procName`. The values in `procParams` specify the list of values to be passed to the PL/SQL procedure.

**PL/SQL Procedure Definition**

The definition of the PL/SQL procedure must adhere to the following guidelines:

- The PL/SQL procedure must be accessible from the SYSMAN schema

- The PL/SQL procedure must have the following signature:

```
PROCEDURE MySQLProc(p_param_names    MGMT_JOB_VECTOR_PARAMS,
                    p_proc_params    MGMT_JOB_VECTOR_PARAMS,
                    p_param_list  OUT MGMT_JOB_PARAM_LIST)
```

The list of parameters specified in `paramNames` are passed as `p_param_names` to the procedure.

The comma-separated list of values specified in `procParams` allows you to pass a list of scalar (string/VARCHAR2) values as parameters to the procedure. These values are substituted with job parameter references (if used), bundled into an array (in the order specified in the XML) and passed to the PL/SQL procedure as the second parameter (`p_proc_params`).

The third parameter is an OUT parameter that contains the list of parameters fetched by the procedure. The names of the parameters returned by this OUT parameter must match the names specified in `p_param_names`.

> **Note:**
>
> Although this check is not currently enforced, Oracle recommends strongly that you ensure that the names of the parameters returned by `p_param_list` matches or is a subset of the list of parameter names passed in `p_param_names`.

**Example**

The following SQL parameter source creates a parameter named db_role_suffix based on an existing parameter named db_role. It also preserves the type (scalar/vector) of the original parameter and therefore looks up the parameter from the internal tables rather than have its value passed (db_role is passed as a literal rather than as a substituted value). The values of job_id and job_execution_id are passed substituted.

```
<paramSource sourceType="sql" paramNames="db_role_suffix">
  <sourceParam name="procName"   value="MGMT_JOB_FUNCTIONS.get_dbrole_
    prefix"/>
  <sourceParam name="procParams" value="%job_id%, %job_execution_id%, db_
    role"/>
</paramSource>
```

Within the PL/SQL procedure MGMT_JOB_FUNCTIONS.get_dbrole_prefix, the p_proc_params list contains the values corresponding to the job_id at index 1 and the execution_id at index 2, while the element at index 3 corresponds to the literal text db_role.

**Available SQL Paramsource Procedures**

The Job System team provided the following PL/SQL procedures for use in job types across Enterprise Manager:

*   is_null

    Checks whether the passed job variable is null. A missing variable is also considered null. For each variable passed, the procedure creates a corresponding variable with the scalar value true if the passed variable is non-existent or null. For all other cases, the scalar value false is set. A vector of zero elements is considered non-null.

    Example:

```
    <paramSource sourceType="sql" paramNames="a_is_null, b_is_null,
c_is_null">
        <sourceParam name="procName"   value="MGMT_JOB_FUNCTIONS.is_null"/>
        <sourceParam name="procParams" value="%job_id%, %job_execution_id%, a,
b,
```

```
        c"/>
    </paramSource>
```

In this example, the job variables a, b, and c are checked for null values and the variables a_is_null, b_is_null, and c_is_null are assigned the values of true or false correspondingly.

- add_dbrole_prefix

  For every variable passed, the procedure prefixes the string AS if the value is not null or Normal (case-insensitive), otherwise it returns null. Therefore, a variable with value SYSDBA results in a value of AS SYSDBA, but a value of Normal returns null. If the passed variable corresponds to a vector, the same logic is applied to each individual element of the vector. This is useful while using DB credentials to connect to a SQL*Plus session.

  **Example**:

  ```
  <paramSource sourceType="sql" paramNames="db_role_suffix1, db_role_
   suffix2">
    <sourceParam name="procName"   value="MGMT_JOB_FUNCTIONS.get_dbrole_
      prefix"/>
    <sourceParam name="procParams" value="%job_id%, %job_execution_id%,
  db_
      role1, db_role2"/>
  </paramSource>
  ```

  Here, the values of the variables db_role1 and db_role2 are prefixed with AS as necessary and saved into variables db_role_suffix1 and db_role_suffix2 respectively.

# About the User Parameter Source

The job system also offers a special parameter source called `"user"`, which indicates that a set of parameters must be supplied when a job of that type is submitted. If a parameter is declared to be of source "user" and the "required" attribute is set to "true", then the job system validates that all specified parameters in the source are provided when a job is submitted.

The user source can be evaluated at job submission time or job execution time. When evaluated at submission time, it causes an exception to be thrown if any required parameters are missing. When evaluated at execution time, it causes the execution to fail or stop if there are any missing required parameters.

```
<paramInfo>
    <!-- Indicate that parameters a, b and c are required params -->
    <paramSource paramNames="a, b, c" required="true" sourceType="user" />
</paramInfo>
```

The user source can also be used to indicate that a pair of parameters are target parameters. For example:

```
<paramInfo>
    <!-- Indicate that parameters a, b, c, d, e, f are target params -->
    <paramSource paramNames="a, b, c, d, e, f" sourceType="user" >
        <sourceParam name="targetNameParams" value="a, b, c" />
        <sourceParam name="targetTypeParams" value="d, e, f" />
    </paramSource>
</paramInfo>
```

This example indicates that parameters (a,d), (b,e), (c,f) are parameters that hold target information. Parameter "a" holds target names and "d" holds the corresponding target types. Similarly with parameters "b" and "e", and "c" and "f". For each parameter that holds target names, there must be a corresponding parameter that holds target types. The parameters can be either scalar or vector.

## About the Inline Parameter Source

The `inline` parameter source allows job types to define parameters in terms of other parameters. It is a convenient mechanism to construct parameters that can be reused in other parts of the job type. The following example creates a parameter called `filename` based on the job execution id, for use in other parts of the job type.

```
<jobType>
     <paramInfo>
          <!-- Indicate that value for parameter filename is provided inline -->
          <paramSource paramNames="fileName" sourceType="inline" >
               <sourceParam name="paramValues" value="%job_execution_id%.log" />
               </paramSource>
     </paramInfo>
.....
     <stepset ID="main" type="serial">
       <step command="putFile" ID="S1">
           ...
            <param name="destFile">%fileName%</param>
           ...
       </step>
     </stepset>
</jobType>
```

The following example sets a vector parameter called vparam to be a vector of the values v1, v2, v3, and v4. Only one vector parameter at a time can be set using the inline source.

```
<jobType>
     <paramInfo>
          <!-- Indicate that value for parameter vparam is provided inline -->
          <paramSource paramNames="vparam" sourceType="inline" >
              <sourceParam name="paramValues" value="v1,v2,v3,v4" />
              <sourceParam name="vectorParams" value="vparam" />
          </paramSource>
     </paramInfo>
....
```

## Using the checkValue Parameter Source

The `checkValue` parameter source enables job types to have the job system check that a specified set of parameters has a specified set of values. If a parameter does not have the specified value, then the job system either terminates or suspends the job.

```
<paramInfo>
     <!-- Check that the parameter halt has the value true. If not, suspend the job
-->
     <paramSource paramNames="halt" sourceType="checkValue" >
         sourceParam name="paramValues" value="true" />
         <sourceParam name="action" value="suspend" />
```

```
            </paramSource>
    </paramInfo>
```

The following example checks whether a vector parameter v has the values v1,v2,v3, and v4. Only one vector parameter at a time can be specified in a checkValue parameter source. If the vector parameter does not have those values, in that order, then the job is terminated.

```
<paramInfo>
    <!-- Check that the parameter halt has the value true. If not, suspend the
job -->
    <paramSource paramNames="v"  sourceType="checkValue" >
        <sourceParam name="paramValues" value="v1,v2,v3,v4" />
        <sourceParam name="action" value="abort" />
        <sourceParam name="vectorParams" value="v" />
    </paramSource>
</paramInfo>
```

## About the properties Parameter Source

The `properties` parameter source fetches a named set of target properties for each of a specified set of targets and stores each set of property values in a vector parameter.

The following example fetches the properties `"OracleHome"` and `"OracleSID"` for the specified set of targets (dlsun966 and ap952sun) into the vector parameters `ohomes` and `osids`, respectively. The first vector value in the `ohomes` parameter will contain the OracleHome property for dlsun966, and the second will contain the OracleHome property for ap952sun. Likewise with the `OracleSID` property.

```
<paramInfo>
    <!-- Fetch the OracleHome and OracleSID property into the vector params
ohmes, osids -->
    <paramSource paramNames="ohomes,osids" overrideUser="true"
sourceType="properties">
        <sourceParams>
            <sourceParam name="propertyNames" value="OracleHome,OracleSID" />
            <sourceParam name="targetNames" value="dlsun966,ap952sun" />
            <sourceParam name="targetTypes" value="host,host" />
        </sourceParams>
    </paramSource>
</paramInfo>
```

As with the credentials source, vector parameter names can be provided for the target names and types.

```
<paramInfo>
    <!-- Fetch the OracleHome and OracleSID property into the vector params
ohmes, osids -->
    <paramSource paramNames="ohomes,osids" overrideUser="true"
sourceType="properties">
        <sourceParams>
            <sourceParam name="propertyNames" value="OracleHome,OracleSID" />
            <sourceParam name="targetNamesParam" value="job_target_names" />
            <sourceParam name="targetTypes" value="job_target_types" />
        </sourceParams>
    </paramSource>
</paramInfo>
```

## Understanding Parameter Sources and Parameter Substitution

Parameter sources are applied in the order they are specified. Parameter substitution (of the form %param%) can be used inside `sourceParam` tags, but the substituted parameter must exist when the parameter source is evaluated. Otherwise, the job system substitutes an empty string in its place.

## About Parameter Encryption

The job system offers the facility of storing specified parameters in encrypted form. Parameters that contain sensitive information, such as passwords, must be stored in encrypted form. A job type can indicate that parameters fetched through a parameter source be encrypted by setting the encrypted attribute to true in a parameter source.

For example:

```
<paramInfo>
    <!-- Fetch params from the credentials table into vector parameters; store
them encrypted -->
    <paramSource paramNames="vec_usernames,vec_passwords" overrideUser="true"
                                sourceType="credentials" encrypted="true">
      <sourceParams>
            <sourceParam name="credentialType" value="patch" />
            <sourceParam name="credentialColumns"
value="node_username,node_password" />
            <sourceParam name="targetNames" value="dlsun966,ap952sun" />
            <sourceParam name="targetTypes" value="host,host" />
            <sourceParam name="credentialScope" value="system" />
      </sourceParams>
    </paramSource>
</paramInfo>
```

A job type can also specify that parameters supplied by the user be stored in encrypted form:

```
<paramInfo>
    <!-- Indicate that parameters a, b and c are required params -->
    <paramSource paramNames="a, b, c" required="true" sourceType="user"
encrypted="true" />
</paramInfo>
```

# Specifying Credential Information

Until Oracle Enterprise 11*g* release 1, credentials were represented as two parameters, (user name and password). The job type owner can either have a credential parameter source to extract these parameters or define these as user parameters, and then pass on the parameters to the various steps that require the parameters.

This required knowledge about the credential set, credential types, and their columns, along with knowledge about various authentication mechanisms, must be supported by the job type, irrespective of the pool of authentication schemes that could be supported by the Enterprise Manager. This restricted the freedom of the job type owner to model just the job type and ignore the authentication required to perform the operations. To overcome these issues and to evolve a unified mechanism in the

job type to specify the credentials, Oracle introduced a new concept called credential usage.

## About Credential Usage

A credential usage is the point where the credential is required to perform an operation. Credential submissions must be made against these usages only.

## Overview of Credential Binding

A credential binding is a reference of a credential by a step. Each step exposes its credential usage which must be fulfilled in the metadata. Therefore, each credential binding refers to a reference credential usage that is defined in the credential usage section of the metadata. When the step requests its own credential usage, a binding helps resolve which credential submission in a particular automation entity (Job or DP instance) must be passed to that step.

In earlier releases, the job types had a credential parameter source to extract the user name and password from the credentials (`JobCredRecord`) passed to the job and then these were available as parameters to the entire job type. This behavior is deprecated with no support and is superseded by the new credential usage structure.

The following Job type example shows the use of credentials declaration in the job type:

```
<jobType version="1.0" name="OSCommandNG"
        singleTarget="true" targetTypes="all"
        defaultTargetType="host" editable="true"
        restartable="true" suspendable="true" >
    <credentials>
        <credential usage="hostCreds" authTargetType="host"
                    defaultCredentialSet="HostCredsNormal"/>
    </credentials>
    <paramInfo>
        <paramSource sourceType="user" paramNames="command"
             required="true" evaluateAtSubmission="true" />
        <paramSource sourceType="inline"
                    paramNames="TargetName,TargetType"
                 overrideUser="true"
           evaluateAtSubmission="true">
            <sourceParam name="paramValues"
                      value="%job_target_names%[1],
                        %job_target_types%[1]" />
        </paramSource>
        <paramSource sourceType="properties"
                    overrideUser="true"
           evaluateAtSubmission="false" >
           <sourceParam name="targetNamesParam"
                      value="job_target_names" />
           <sourceParam name="targetTypesParam"
                      value="job_target_types" />
        </paramSource>
        <paramSource sourceType="substValues"
                    paramNames="host_command,host_args,os_script"
                 overrideUser="true" evaluateAtSubmission="false">
            <sourceParam name="sourceParams"
                        value="command,args,os_script" />
        </paramSource>
```

```
            </paramInfo>
            <stepset ID="main" type="serial" >
               <step ID="Command" command="sampleRemoteOp">
                  <credList>
                     <cred usage="OS_CRED" reference="hostCreds"/>
                  </credList>
                  <paramList>
                     <param name="remoteCommand">%host_command%</param>
                     <param name="args">%host_args%</param>
                     <param name="input"><![CDATA[%os_script%]]></param>
                     <param name="largeInputParam">large_os_script</param>
                     <param name="substituteLargeParam">true</param>
                     <param name="targetName">%job_target_names%[1]</param>
                     <param name="targetType">%job_target_types%[1]</param>
                     <param name="executeSynchronous">false</param>
                  </paramList>
               </step>
            </stepset>
</jobType>
```

The first set of three lines declares a credential usage in the job type. The next set of lines binds the credential usage to that of the step. The user name and password cannot be extracted by the jobs system and therefore can no longer be exposed as parameters.

# XSD Elements – Credential Usage and Credential Binding

The XSD element credential usage and credentials binding are explained in Table 8-3 and Table 8-4.

**Table 8-3    Credential Usage (credential)**

| Attribute | Required (Y/N) | Description |
|---|---|---|
| usage | Y | Name of the credential through which it will be referred in the job type. All credential submissions are to be made for this name. |
| authTargetType | Y | Target type against which authentication is to be performed for any operation. For example, running "ls" any target means authentication against the host. |
| defaultCredentialSet | Y | Name of the credential set to be picked up as a credential if no submissions are found for the credential usage when required. |
| credentialTypes | N | Name of the credential types which can only be used for specifying the credentials. This is to facilitate filtering of credentials in the credential selector UI component. |
| displayName | N | Name that is intended to be shown in the credential selector UI. |
| description | N | Description that is intended to be shown in the credential selector UI. |

**Table 8-4    Credential Binding (cred)**

| Attribute / sub element | Required (Y/N) | Description |
|---|---|---|
| usage | Y | Credential usage understood by the step. |
| reference | Y | Credential usage referred to and present in the declarations of the job type or DP metadata. |

> **Note:**
>
> The Credential Binding element can only be used inside the step or job elements in the job type XML.

# Specifying Security Information

Typically, a job type tends to perform actions that can be considered to be "privileged". For example, patching a production database or affecting the software installed in an Oracle home directory or the APPL_TOP directory. Such job types must be submitted by Enterprise Manager users that have the appropriate level of privileges to perform these actions.

The job system provides a section called securityInfo, which the author of a job type can use to specify the minimum level of privileges (system and target) that the submitter of a job of this type must have.

The securityInfo section enables the job type author to encapsulate the security requirements associated with submitting a job in the job type itself. No further code must be written to enforce security. Also, it ensures that Enterprise Manager users cannot directly submit jobs of a specific type (using the job system APIs and bypassing the application) unless they have the set of privileges defined by the job type author.

**Example 1**

The following example shows what a typical securityInfo section looks like. Suppose you are writing a job type that clones a database. This job type requires two targets, a source database and a destination node on which the destination database will be created. This job type requires that the user submitting a clone job have a CLONE FROM privilege on the source (database) and a MAINTAIN privilege on the destination (node).

In addition, the user requires the CREATE TARGET system privilege to introduce a new target into the system. Assuming that the job type is written so that the first target in the target list is the source and the second target in the target list is the destination, the security requirements for such a job type could be addressed as follows:

```
<jobType>
  <securityInfo>
    <privilege name="CREATE TARGET" type="system" />
    <privilege name="CLONE FROM" type="target" evaluateAtSubmission="false" >
        <target name="%job_target_names%[1]" type="%job_target_types%[1]" />
    </privilege>
    <privilege name="MAINTAIN" type="target" evaluateAtSubmission="false">
```

```
            <target name="%job_target_names%[2]" type="%job_target_types%[2]" />
      </privilege>
  </securityInfo>
  <!-- An optional <paramInfo> section will follow here, followed by the stepset
      definition of the job
  -->
  <paramInfo>
   ....
  </paramInfo>
  <stepset ...>
  </stepset>
</jobType>
```

The securityInfo section is a set of `<privilege>` tags. Each privilege could be a system or target privilege, as indicated by the type attribute of the tag. If the privilege is a target privilege, then the targets that the privilege is attached to must be explicitly enumerated, or else the target_names_param and target_types_param attributes must be used as shown in the following example. The usual %param% notation can be used to indicate job parameter and target placeholders.

By default, all `<privilege>` directives in the securityInfo section are evaluated at job submission time, after all submit-time parameter sources have been evaluated. The job system throws an exception if the user does not have any of the privileges specified in the securityInfo section.

Execution-time parameter sources are not evaluated at job submission time, so take care not to use job parameters that might not have been evaluated yet. You could also direct the job system to evaluate a privilege directive at job execution time by setting the evaluateAtSubmission parameter to false.

The only reason you might want to do this is if the exact set of targets that the job is operating on is unknown until the job execution time (for example, it is computed using an execution-time parameter source). Execution-time privilege directives are evaluated after all execution-time parameter sources are evaluated.

**Example 2**

Assume that you are writing a job type that requires a MODIFY privilege on each one of its targets, but the exact number of targets is unknown at the time of writing. Use the target_names_param and target_types_param attributes for this purpose. These specify vector parameters from which the job system will get the target names and the corresponding target types. These could be any vector parameters. This example uses the job target list (job_target_names and job_target_types).

```
<securityInfo>
    <privilege name="MODIFY" type="target" target_names_param="job_target_names"
                target_types_param="job_target_types" />
</securityInfo>
```

# Specifying Lock Information

Often executing jobs need to acquire resources. For example, a job applying a patch to a database might need a mechanism to ensure that other jobs (submitted by other users in the system) on the database are prevented from running while the patch is being applied. In other words, it might want to acquire a lock on the database target so that other jobs that try to acquire the same lock block (or terminate). This allows a patch job, once it starts, to perform its work without disruption.

Sometimes, locks could be at more than one level. A hot backup of a database, for example, can allow other hot backups to proceed (because they do not bring down the database), but cannot allow cold backups or database shutdown jobs to proceed (because they shut down the database, causing the backup to fail).

A job execution indicates that it is reserving a resource on a target by acquiring a lock on the target. A lock is a proxy for reserving some part of the functionality of a target. When an execution acquires a lock, it blocks other executions that try to acquire the same lock on the target. A lock is identified by a name and a type and can be of the following types:

- Global: These are locks that are not associated with a target. An execution that holds a global lock blocks other executions that are trying to acquire the same global lock (such as a lock with the same name).

- Target Exclusive: These are locks that are associated with a target. An execution that holds an exclusive lock on a target blocks executions that are trying to acquire any named lock on the target, as well as executions trying to acquire an exclusive lock on the target. Target exclusive locks have no name: there is exactly one exclusive lock per target.

- Target Named: A named lock on a target is analogous to obtaining a lock on one particular functionality of the target. A named lock has a user-specified name. An execution that holds a named lock blocks other executions that are trying to acquire the same named lock, as well as executions that are trying to acquire an exclusive lock on the target.

**Example**

Locks that a job type wants to acquire can be obtained by specifying a lockInfo section in the job type. This example lists the locks that the job is to acquire, the types of locks, as well as the targets on which it wants to acquire the locks:

```
<lockInfo action="suspend">
    <lock type="targetExclusive">
        <targetList>
            <target name="%backup_db%" type="oracle_database" />
        </targetList>
    </lock>
    <lock type="targetNamed" name="LOCK1" >
        <targetList>
            <target name="%backup_db%" type="oracle_database" />
            <target name="%job_target_names%[1]" type="%job_target_types%[1]" />
            <target name="%job_target_names%[2]" type="%job_target_types%[2]" />
        </targetList>
    </lock>
    <lock type="global" name="GLOBALLOCK1" />
</lockInfo>
```

This example shows a job type that acquires a target-exclusive lock on a database target whose name is given by the job parameter backup_db. It also acquires a named target lock named "LOCK1" on three targets, namely, the database whose name is stored in the job parameter backup_db, and the first two targets in the target list of the job. Finally, it acquires a global lock named "GLOBALLOCK1". The "action" attribute specifies what the job system should do to the execution if any of the locks in the section cannot be obtained (because some other execution is holding them). Possible values are suspend (all locks are released and the execution state changes to "Suspended:Lock") and abort (the execution terminates). The following points can be made about executions and locks:

- An execution can only attempt to obtain locks when it starts (although it is possible to override this by using nested jobs).

- An execution can acquire multiple locks. Locks are always acquired in the order specified. Because of this, executions can potentially deadlock each other if they attempt to acquire locks in the wrong order.

- Target locks are always acquired on targets in the same order as they are specified in the `<targetList>` tag.

- If a target in the target list is null or does not exist, the execution terminates.

- If an execution attempts to acquire a lock it already holds, it succeeds.

- If an execution cannot acquire a lock (usually because another execution is holding it), it has a choice of suspending itself or terminating. If it chooses to suspend itself, all locks it has acquired so far are released, and the execution is put in the Suspended/Lock state.

- All locks held by an execution are released when an execution finishes (whether it completes, fails, or is stopped). There might be several waiting executions for each released lock and these are sorted by time, with the earliest request getting the lock.

When jobs that have the lockInfo section are nested inside each other, the nested job's locks are obtained when the nested job first executes, not when an execution starts. If the locks are not available, the parent execution can be suspended or terminated, possibly after a few steps have executed already.

**lockInfo Example 1**

In this example, two job types called HOTBACKUP and COLDBACKUP perform hot backups and cold backups, respectively, on the database. The difference is that the cold backup brings the database down, but the hot backup leaves it up. Only one hot backup can execute at a time and it keeps out other hot backups as well as cold backups.

When a cold backup is executing, no other job type can execute (since it shuts down the database as part of its execution). A third job type called SQLANALYZE performs scheduled maintenance activity that results in modifications to database tuning parameters (two SQLANALYZE jobs cannot run at the same time).

Table 8-5 shows the incompatibilities between the job types. An 'X' indicates that the job types are incompatible. An 'OK' indicates that the job types are compatible.

**Table 8-5    Job Type Incompatibilities**

| Job Type | HOTBACKUP | COLDBACKUP | SQLANALYZE |
|---|---|---|---|
| HOTBACKUP | X | X | OK |
| COLDBACKUP | X | X | X |
| SQLANALYZE | OK | X | X |

The following code example shows the lockInfo sections for the three job types. The cold backup obtains an exclusive target lock on the database. The hot backup job does not obtain an exclusive lock, but only the named lock "BACKUP_LOCK". Likewise, the SQLANALYZE job obtains a named target lock called "SQLANALYZE_LOCK".

Assuming that the database that the jobs operate on is the first target in the target list of the job, the lock sections of the two jobs look as follows:

```
<jobType name="SQLANALYZE">
    <lockInfo action="abort">
        <lock type="targetNamed" name="SQLANALYZE_LOCK" >
            <targetList>
               <target name="%job_target_names%[1]"
type="%job_target_names%[1]" />
            </targetList>
        </lock>
    </lockInfo>
    ........ Rest of the job type follows
</jobType>
```

Since a named target lock blocks all target exclusive locks, executing hot backups suspends cold backups, but not analyze jobs (because they try to acquire different named locks). Executing SQL analyze jobs terminates other SQL analyze jobs and suspends cold backups, but not hot backups. Executing cold backups suspends hot backups and terminates SQL analyze jobs.

**lockInfo Example 2**

A job type called PATCHCHECK periodically checks a patch stage area and downloads information about newly staged patches into the Management Repository. Two such jobs cannot run at the same time; however, the job is not associated with any target. The solution is for the job type to attempt to grab a global lock:

```
<jobType name="PATCHCHECK">
    <lockInfo>
        <lock type="global" name="PATCHCHECK_LOCK" />
    </lockInfo>
    ........ Rest of the job type follows
</jobType>
```

**lockInfo Example 3**

A job type that nests the SQLANALYZE type within itself is shown in the following example. The nested job executes after the first step (S1) executes.

```
<jobType name="COMPOSITEJOB">
    <stepset ID="main" type="serial">
        <step ID="S1" ...>
           ....
        </step>
        <job name="nestedsql" type="SQLANALYZE">
           ....
        </job>
    </stepset>
</jobType>
```

In the previous example, the nested job tries to acquire locks when it executes (because the SQLANALYZE has a lockInfo section). If the locks are currently held by other executions, then the nested job terminates (as specified in the lockInfo), which in turn terminates the parent job.

# Suspending a Job or Step

Suspended is a special state that indicates that steps in the job will not be considered for scheduling and execution. A step in an executing job can suspend the job, through the suspend_job PL/SQL API. This suspends both the currently executing step, and the job itself.

Suspending a job means that all steps in the job that are currently in a "scheduled" state are marked as "suspended" and will thereafter not be scheduled or executed. All currently executing steps (for example, parallel stepsets) continue to execute. However, when any currently executing step completes, the next steps in the job will not be scheduled. Instead they are put in suspended state. When a job is suspended on submission, the previous applies to the first steps in the job that would have been scheduled.

Suspended jobs may be restarted at any time by calling the restart_job() PL/SQL API. However, jobs that are suspended because of serialization (locking) rules are not restartable manually. The job system restarts such jobs automatically when currently executing jobs of that job type complete. Restarting a job effectively changes the state of all suspended steps to scheduled and job execution proceeds normally.

# Restarting a Job

If a job is suspended, failed, or terminated, you can restart it from any given step (typically, the stepset that contains a failed or terminated step). For failed or terminated jobs, the steps that get scheduled again depends on which step from which the job is restarted.

## Restarting Versus Resubmitting

If a step in a job is resubmitted, it means that it executes regardless of whether the original execution of the step completed or failed. If a stepset is resubmitted, then the first step, stepset, or job in the stepset is resubmitted, recursively. Therefore, when a job is resubmitted, the entire job is executed again by recursively resubmitting its initial stepset. The parameters and targets used are the same that were used when the job was first submitted. Essentially, the job executes as if it were submitted for the first time with the specified set of parameters and targets. Also, you can use the resubmit_job API in the mgmt_jobs package to resubmit a job. You can resubmit jobs even if the earlier executions completed successfully.

Restarting a job generally refers to resuming job execution from the last failed step (although the job type can control this behavior using the restartMode attribute of steps/stepsets/jobs). Usually, steps from the failed job execution that succeeded are not executed again.

To restart a failed or terminated job, call the restart_job API in the mgmt_jobs package. You cannot restart a job that completed successfully.

## Default Restart Behavior

Restarting a job creates a new execution called the restart execution. The original failed execution of the job is called the source execution. All parameters and targets are copied over from the source execution to the restart execution. Parameter sources

are not reevaluated, unless the original job terminated because of a parameter source failure.

To restart a serial or iterative stepset, the job system first examines the status of the serial stepset. If the status of the serial stepset is "Completed", then all the entries for its constituent steps are copied over from the source execution to the restart execution. If the status of the stepset is "Failed" or "Aborted", then the job system starts top down from the first step in the stepset.

If the step previously completed successfully in the source execution, it is copied to the restart execution. If the step previously failed or aborted, it is rescheduled for execution in the restart execution. After this step has finished executing, the job system determines the next steps to execute. These could be successOf or failureOf dependencies, or simply steps/stepsets/jobs that execute after the current step.

If the subsequent step completed successfully in the source execution, then it will not be scheduled for execution again and the job system copies the source execution status to the restart execution for that step. It continues in this fashion until it reaches the end of the stepset. It then recomputes the status of the stepset based on the new executions.

To restart a parallel stepset, the job system first examines the status of the parallel stepset. If the status of the stepset is "Completed", then all the entries for its constituent steps are copied over from the source execution to the restart execution. If the status of the stepset is "Failed" or "Aborted", then the job system copies over all successful steps in the steps from the source to the restart execution. It reschedules all steps that failed or terminated in the source execution, in parallel. After these steps have finished executing, the status of the stepset is recomputed.

To restart a nested job, the restart algorithm is applied recursively to the first (outer) stepset of the nested job.

In the previous paragraphs, if one of the entities is a stepset or a nested job, then the restart mechanism is applied recursively to the stepset or job. When entries for steps are copied over to the restart execution, the child execution entries point to the same output Character Large Object (CLOB) entries as the parent execution.

## Using the restartMode Directive

A job type can affect the restart behavior of each step, stepset, or job within it by the use of the restartMode attribute. You can set this to "failure" (default) or "always".

- When set to failure and the top-down copying process described in the previous section occurs, the step, stepset, or job is copied without being executed again if it succeeded in the source execution. If it failed or terminated in the source execution, then it restarts recursively at the last point of failure.

- When the restartMode attribute is set to "always" for a step, the step is always executed again in a restart, regardless of whether it succeeded or failed in the source execution. The use of this attribute is useful when certain steps in a job must always be executed again in a restart (for example, a step that shuts down a database before backing it up).

For a stepset or nested job, if the restartMode attribute is set to "always", then all steps in the stepset/nested job are restarted, even if they completed successfully in the source execution. If it is set to "failure", then restart is attempted only if the status of the stepset or nested job was set to Failed or Aborted in the source execution.

Individual steps inside a stepset or nested job might have their restartMode set to "always" and such steps are always executed again.

**Restart Examples**

The following sections discuss a range of scenarios related to restarting stepsets.

**Example 1**

Consider the serial stepset with the sequence of steps below:

```
<jobtype ...>
<stepset ID="main" type="serial" >
    <step ID="S1" ...>
     ...
    </step>
    <step ID="S2" ...>
     ...
    </step>
    <step ID="S3" failureOf="S2"...>
     ...
    </step>
    <step ID="S4" successOf="S2"...>
     ...
    </step>
</stepset>
</jobtype>
```

In this stepset, assume the source execution had S1 execute successfully and step S2 and S3 (the failure dependency of S2) fail.

When the job is restarted, step S1 is copied to the restart execution from the source execution without being re-executed (because it successfully completed in the source execution). Step S2, which failed in the source execution, is rescheduled and executed.

If S2 completes successfully, then S4, its success dependency (which never executed in the source execution), is scheduled and executed. The status of the stepset (and the job) is the status of S4.

If S2 fails, then S3 (its failure dependency) is rescheduled and executed (since it had failed in the source execution), and the status of the stepset (and the job) is the status of S3.

Assume that step S1 succeeded, S2 failed, and S3 (its failure dependency) succeeded in the source execution. As a result, the stepset (and therefore the job execution) succeeded. This execution cannot be restarted because the execution completed successfully although one of its steps failed.

Finally, assume that steps S1 and S2 succeed, but S4 (S2's success dependency) failed. S3 is not scheduled in this situation. When the execution is restarted, the job system copies over the executions of S1 and S2 from the source to the restart execution, and reschedules and executes S4. The job succeeds if S4 succeeds.

**Example 2**

Consider the following:

```
<jobtype ...>
<stepset ID="main" type="serial" stepsetStatus="S2" >
    <step ID="S1" restartMode="always" ...>
```

```
      ...
     </step>
     <step ID="S2" ...>
      ...
     </step>
     <step ID="S3" ...>
      ...
     </step>
   </stepset>
</jobtype>
```

In the previous example, assume that step S1 completes and S2 fails. S3 executes (because it does not have a dependency on S2) and succeeds. The job, however, fails, because the stepset main has its stepsetStatus set to S2.

When the job is restarted, S1 is executed again, although it completed the first time, because the restartMode of S1 was set to "always".

Step S2 is rescheduled and executed, because it failed in the source execution. After S2 executes, step S3 is *not* rescheduled for execution again, because it executed successfully in the source execution. If the intention is that S3 must execute in the restart execution, then its restartMode must be set to "always".

In the previous example, if S1 and S2 succeeded and S3 failed, the stepset main would still succeed (because S2 determines the status of the stepset). In this case, the job succeeds, and cannot be restarted.

**Example 3**

Consider the following example:

```
<jobtype ...>
<stepset ID="main" type="serial"  >
  <stepset type="serial" ID="SS1" stepsetStatus="S1">
    <step ID="S1" ...>
     ...
    </step>
    <stepset ID="S2" ...>
     ...
    </step>
  </stepset>
  <stepset type="parallel" ID="PS1" successOf="S1" >
    <step ID="P1" ...>
     ...
    </step>
    <step ID="P2" ...>
     ...
    </step>
    <step ID="P3" ...>
     ...
    </step>
  </stepset>
</stepset>
</jobtype>
```

In this example, assume that steps S1 and S2 succeeded (and therefore, stepset SS1 completed successfully). Thereafter, the parallel stepset PS1 was scheduled, and assume that P1 completed, but P2 and P3 failed. As a result, the stepset "main" (and the job) failed.

When the execution is restarted, the steps S1 and S2 (and therefore the stepset SS1) are copied over without execution. In the parallel stepset PS1, both the steps that failed (P2 and P3) are rescheduled and executed.

Assume that S1 completed and S2 failed in the source execution. Stepset SS1 still completed successfully because the status of the stepset is determined by S1, not S2 (because of the stepsetStatus directive). Assume that PS1 was scheduled and P1 failed, and P2 and P3 executed successfully. When this job is rescheduled, the step S2 will not be executed again (because the stepset SS1 completed successfully). The step P1 is not rescheduled and executed.

**Example 4**

Consider a slightly modified version of the XML in "Example 3":

```
<jobtype ...>
<stepset ID="main" type="serial"  >
  <stepset type="serial" ID="SS1" stepsetStatus="S1" restartMode="always" >
    <step ID="S1" ...>
     ...
    </step>
    <stepset ID="S2" ...>
     ...
    </step>
  </stepset>
  <stepset type="parallel" ID="PS1" successOf="S1" >
    <step ID="P1" ...>
     ...
    </step>
    <step ID="P2" ...>
     ...
    </step>
    <step ID="P3" ...>
     ...
    </step>
  </stepset>
</stepset>
</jobtype>
```

In the previous example, assume that S1 and S2 succeeded (and therefore, stepset SS1 completed successfully). Thereafter, the parallel stepset PS1 was scheduled, and assume that P1 completed, but P2 and P3 failed. When the job is restarted, the entire stepset SS1 is restarted (since the restartMode is set to "always"). This means that steps S1 and S2 are successively scheduled and executed. Now the stepset PS1 is restarted, and because the restartMode is not specified (it is always "failure" by default), it is restarted at the point of failure, which in this case means that the failed steps P2 and P3 are executed again, but not P1.

# Adding Job Types to the Job Activity and Job Library Pages

To make a new job type accessible from the Enterprise Manager Cloud Console **Job Activity** or **Job Library** page, you must to modify the following specific XML tag attributes.

- To display the job type on Job Activity page, set useDefaultCreateUI to "true" as shown in the following example.

  ```
  <displayInfo useDefaultCreateUI="true"/>
  ```

- To display the job type on the Job Library page, in addition to setting useDefaultCreateUI attribute, you must also set the jobtype editable attribute to "true."

```
<jobtype name="jobType1" editable="true">
```

If you set useDefaultCreateUI="true" and editable="false", then the job type appears on the Job Activity page only and not on Job Library page.This means you cannot edit the job definition.

## Adding a Job Type to the Job Activity Page

Figure 8-1 shows the result of setting the useDefaultCreateUI attribute to "true" and enabling users to create a job to select the newly added job type from the Create Job menu.

**Figure 8-1    Available Job Types from the Job Activity Page**
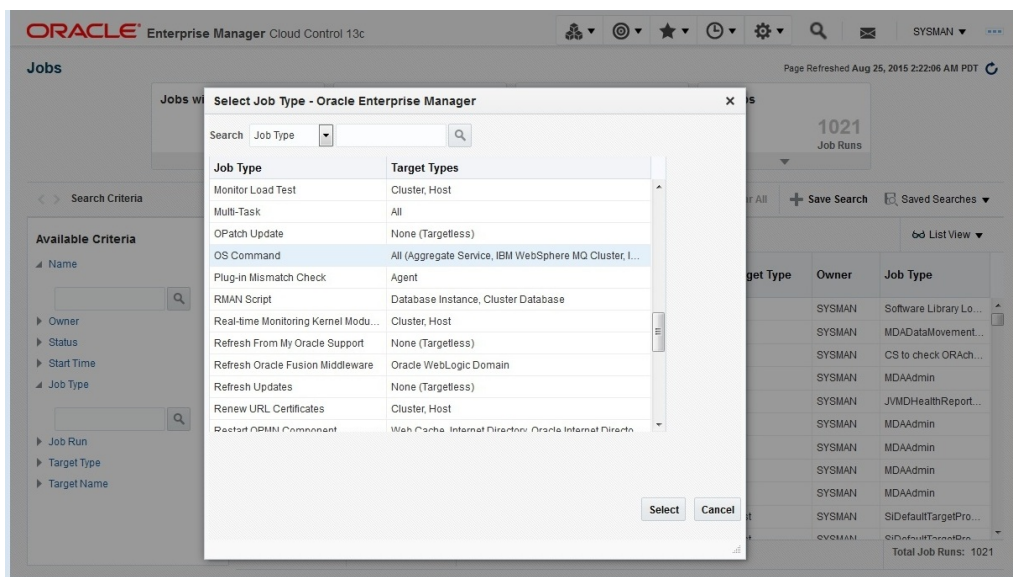


Making the job type available from the **Job Activity** page also permits access to the default Create Job user interface when a user attempts to create a job using the newly added job type.

## Adding the displayInfo Tag

You can add the displayInfo tag to the job definition file at any point after the </stepset> tag and before the </jobtype> tag at the end of the job definition file, as shown in the following example.

```
<jobtype ...>
<stepset ID="main" type="serial"  >
  <stepset type="serial" ID="SS1" stepsetStatus="S1">
    <step ID="S1" ...>
     ...
    </step>
    <stepset ID="S2" ...>
     ...
```

```
        </step>
      </stepset>
      <stepset type="parallel" ID="PS1" successOf="S1" >
        <step ID="P1" ...>
         ...
        </step>
        <step ID="P2" ...>
         ...
        </step>
        <step ID="P3" ...>
         ...
        </step>
      </stepset>
    </stepset>
    <displayInfo useDefaultCreateUI="true"/>
    </jobtype>
```

## Adding a Job Type to the Job Library Page

To make the job type available from the Job Library page, you must also set the
jobType tag's editable attribute to "true" in addition to adding the displayInfo tag,
This makes the newly added job type a selectable option from the Create Library Job
menu.

## Making the Job Type Editable

The editable attribute of the jobtype tag is set at the beginning of the job definition file,
as shown in the following example.

```
<jobtype name="jobType1" editable="true">
<stepset ID="main" type="serial"  >
  <stepset type="serial" ID="SS1" stepsetStatus="S1">
    <step ID="S1" ...>
     ...
    </step>
    <stepset ID="S2" ...>
     ...
    </step>
  </stepset>
  <stepset type="parallel" ID="PS1" successOf="S1" >
    <step ID="P1" ...>
     ...
    </step>
    <step ID="P2" ...>
     ...
    </step>
    <step ID="P3" ...>
     ...
    </step>
  </stepset>
</stepset>
<displayInfo useDefaultCreateUI="true"/>
</jobtype>
```

# Examples: Specifying Job Types in XML

The following sections provide examples of specifying job types in XML.

**Example 1**

This example describes a job type called jobType1 that defines four steps, S1, S2, S3, and S4. It executes S1 and S2 serially, one after another. It executes step S3 *only* if step S2 succeeds, and step S4 *only* if S2 fails. All the steps execute within an iterative subset, so these actions are performed in parallel on all targets in the job target list of type database.

> **Note:**
>
> These examples use percentage (%) symbols to indicate parameters, %patchno%, %username%, %password%, and %job_target_name%.
>
> The job system substitutes the value of a job parameter named "patchno" in place of the %patchno%. Likewise, it substitutes the values of the corresponding parameters for %username% and %password%. %job_target_name% and %job_target_type% are "pre-built" placeholders that substitute the name of the target that the step is currently executing against.

The steps S2, S3, and S4 illustrate how you can use the remoteOp command to execute a SQL*Plus script on the Management Agent.

The status of a job is failed if any of the following occurs:

*   S2 fails and S4 fails

*   S2 succeeds and S3 fails

Because S2 executes after S1 (regardless of whether S1 succeeds or fails), the status of S1 does not affect the status of the job.

**Example: Job Type Defining Four Steps**

```
<jobtype name="jobType1" editable="true" version="1.0">
<credentials>
    <credential usage="defaultHostCred" authTargetType="host"
           defaultCredentialSet="DBHostCreds"/>
    <credential usage="defaultDBCred" authTargetType="oracle_database"
           credentialTypes="DBCreds"
           defaultCredentialSet="DBCredsNormal"/>
    </credentials>
    <stepset ID="main" type="iterativeParallel" iterate_param="job_target_types"
iterate_param_filter="oracle_database" >
    <step ID="s1" command="remoteOp"">
    <credList>
  <cred usage="defaultHostCred" reference="defaultHostCred"/>
    </credList>
    <paramList>
      <param name="remoteCommand">myprog</param>
      <param name="targetName">%job_target_names%[%job_iterate_
       index%]
         </param>
        <param name="targetType">%job_target_types%[%job_iterate_
         index%]
         </param>
         <param name="args">-id=%patchno%</param>
         <param name="successStatus">3</param>
```

```
          <param name="failureStatus">73</param>
        </paramList>
      </step>
      <step ID="s2" command="remoteOp"">
        <credList>
         <cred usage="defaultHostCred" reference="defaultHostCred"/>
        </credList>
        <paramList>
          <param name="remoteCommand">myprog2</param>
          <param name="targetName">%job_target_names%[%job_iterate_
           index%]</param>
          <param name="targetType">%job_target_types%[%job_iterate_
           index%]</param>
          <param name="args">-id=%patchno%</param>
          <param name="successStatus">3</param>
          <param name="failureStatus">73</param>
        </paramList>
      </step>
      <step ID="s3" successOf="s2" command="remoteOp">
      <credList>
       <cred usage="defaultHostCred" reference="defaultHostCred"/>
       <cred usage="defaultDBCred" reference="defaultDBCred">
          <map toParam="db_username" credColumn="DBUserName"/>
          <map toParam="db_passwd" credColumn="DBPassword"/>
          <map toParam="db_alias" credColumn="DBRole"/>
      </cred>
      </credList>
        <paramList>
          <param name="command">prog1</command>
          <param name="script">
          <![CDATA[
            select * from MGMT_METRICS where target_name=%job_target_type%
[%job_
            iterate_param_index%]
          ]]>
          </param>
        <param name="args">%db_username%/%db_passwd%@%db_alias%</param>
        <param name="targetName">%job_target_names%[%job_iterate_
         index%]</param>
        <param name="targetType">%job_target_types%[%job_iterate_
         index%]</param>
        <param name="successStatus">0</param>
        <param name="failureStatus">1</param>
        </paramList>
      </step>
           <step ID="s4" failureOf="s2" command="remoteOp">
      <credList>
        <cred usage="defaultHostCred" reference="defaultHostCred"/>
      </credList>
      <paramList>
      <param name="input">
      <![CDATA[
         This is standard input to the executed progeam. You can use
placeholders
         for parameters, such as
         %job_target_name%[%job_iterate_param_index%]
      ]]>
      </param>
      <param name="remoteCommand">prog2</param>
      <param name="targetName">%job_target_names%[%job_iterate_
       index%]</param>
```

```
        <param name="targetType">%job_target_types%[%job_iterate_

index%]</param>
    <param name="args"></param>
    <param name="successStatus">0</param>
    <param name="failureStatus">1</param>
   </paramList>
  </step>
  </stepset>
<displayInfo useDefaultCreateUI="true"/>
</jobtype>
```

**Example 2**

This example describes a job type that has two steps, S1 and S2, that execute in parallel (within a parallel stepset ss1) and a third step, S3, that executes *only* after both S1 and S2 have completed successfully. This is achieved by placing the step S3 in a serial stepset ("main") that also contains the parallel stepset ss1. This job type is a "multi-node" job. The example uses %job_target_name%[1], %job_target_name%[2] in the parameters to the commands. In stepsets other than an iterative stepset, you can only refer to job targets by using their position in the targets array (which is ordered).

%job_target_name%[1] refers to the first target, %job_target_name%[2] to the second, and so on. The assumption is that most multi-node jobs expect their targets to be in some order. For example, a clone job might expect the source database to be the first target, and the target database to be the second target. This job fails if any of the following occurs:

*   The parallel stepset SS1 fails (either S1, or S2, or both fail)

*   Both S1 and S2 succeed, but S3 fails

The job type has declared itself to be Agent-bound. This means that the job is set to Suspended/Agent Down state if either Management Agent (corresponding to the first target or the second target) goes down.

**Example: Job Type Defining Two Steps Followed by a Third Step**

```
<jobtype name="jobType2" version="1.0" agentBound="true" >
  <stepset ID="main" type="serial" editable="true">
      <!-- All steps in this stepset ss1 execute in parallel -->
    <credentials>
    <credential usage="hostCreds" authTargetType="host"
          defaultCredentialSet="HostCredsNormal"/>
    </credentials>
    <stepset ID="ss1" type="parallel" >
      <step ID="s1" command="remoteOp" >
      <credList>
        <cred usage="defaultHostCred" reference="defaultHostCred"/>
      </credList>
        <paramList>
          <param name="remoteCommand">myprog</param>
          <param name="targetName">%job_target_names%[1]</param>
          <param name="targetType">%job_target_types%[1]</param>
          <param name="args">-id=%patchno%</param>
          <param name="successStatus">3</param>
          <param name="failureStatus">73</param>
        </paramList>
      </step>
    <step ID="s2" command="remoteOp" >
    <credList>
```

```
        <cred usage="defaultHostCred" reference="hostCreds"/>
        </credList>
            <paramList>
              <param name="remoteCommand">myprog</param>
              <param name="targetName">%job_target_names%[2]</param>
              <param name="targetType">%job_target_types%[2]</param>
              <param name="args">-id=%patchno%</param>
              <param name="successStatus">3</param>
              <param name="failureStatus">73</param>
            </paramList>
          </step>
        </stepset>
        <!-- This step executes after stepset ss1 has executed, since it is inside the
serial subset "main"
        -->
        <step ID="s3" successOf="ss1" command="remoteOp" >
          ...
        </step>
      </stepset>
<displayInfo useDefaultCreateUI="true"/>
</jobtype>
```

**Example 3**

This example defines a new job type called jobType3 that executes jobs of type jobType1 and jobType2 consecutively. The job2 job of type jobType2 is executed only if the first job fails. To execute another job, the target list and the param list must be passed. The `targetList` tag has a parameter called allTargets, which when set to true, passes along the entire target list passed to this job. By setting allTargets to false, a job type has the option of passing along a subset of its targets to the other job type.

In this example, jobType3 passes along all its targets to the instance of the job of type jobType1, but only the first two targets in its target list (in that order) to the job instance of type jobType2. There is another attribute called allParams (associated with paramList) that performs a similar function with respect to parameters. If allParams is set to true, then all parameters of the parent job are passed to the nested job. Typically the nested job has a different set of parameters (with different names).

If allParams is set to false (default), then the job type can name the nested job parameters explicitly and they do not have to have the same names as those in the parent job. Use parameter substitution to express the nested job parameters in terms of the parent job parameters, as shown in this example.

You can express the dependencies between nested jobs just as if they were steps or stepsets. In this example, a job of type jobType3 succeeds if either:

- the nested job job1 succeeds
- job1 fails and job2 succeeds

**Example: Defining a Job Type That Executes Jobs of Other Job Types**

```
<jobType name="jobType3" editable="true" version="1.0">
  <stepset ID="main" type="serial">
    <job type="jobType1" ID="job1" >
      <target_list allTargets="true" />
      <paramList>
        <param name="patchno">%patchno%</param>
      </paramList>
    </job>
    <job type="jobType2" ID="job2" failureOf="job1" >
```

```
        <targetList>
          <target name="%job_target_names%[1]" type="%job_target_types%[1]" />
          <target name="%job_target_names%[2]" type="%job_target_types%[2]" />
        </targetList>
        <paramList>
          <param name="patchno">%patchno%</param>
        </paramList>
      </job>
    </stepset>
<displayInfo useDefaultCreateUI="true"/>
</jobType>
```

**Example 4**

The Defining a Job Type That Generates Variables in a File example illustrates the use of the `generateFile` command. Assume that you are executing a sequence of scripts, all of which must source a common file that sets up some environment variables, which are known only at runtime. One way to do this is to generate the variables in a file with a unique name. All subsequent scripts are passed this file name as one of their command-line arguments, which they read to set the required environment or shell variables.

The first step, S1, in this job uses the generateFile command to generate a file named *app-home*/*execution-id*.env. Because the execution id of a job is always unique, this ensures a unique file name. It generates three environment variables, ENVVAR1, ENVVAR2, and ENVVAR3, which are set to the values of the job parameters param1, param2 and param3, respectively. These parameters must be set to the right values when the job is submitted.

%job_execution_id% is a placeholder provided by the job system, while %app-home% is a job parameter which must be explicitly provided when the job is submitted.

The second step, S2, executes a script called myscript. The first command-line argument to the script is the generated file name. This script must "source" the generated file, which sets the required environment variables, and then performs its other tasks, as shown in the following code:

```
#!/bin/ksh
ENVFILE=$1
# Execute the generated file, sets the required environment vars
. $ENVFILE
# I can now reference the variables set in the file doSomething $ENVVAR1
$ENVVAR2 $ENVVAR3...
```

The following example provides the full job type specification. Step S3 removes the file that was created by the first step S1. It is important to clean up when using the putFile and generateFile commands to write temporary files on the Management Agent. This example performs the cleanup explicitly as a separate step, but it could also be done by one of the scripts that executes on the remote host.

Additionally, the securityInfo section that specifies the user that submits a job of this job type, must have MAINTAIN privilege on both the targets on which the job operates.

**Example: Defining a Job Type That Generates Variables in a File**

```
<jobtype name="jobType4" editable="true" version="1.0">
  <securityInfo>
    <privilege name="MAINTAIN" type="target" evaluateAtSubmission="false">
      <target name="%job_target_names%[1]" type="%job_target_types%[1]" />
      <target name="%job_target_names%[2]" type="%job_target_types%[2]" />
```

```
        </privilege>
      </securityInfo>
      <credentials>
        <credential usage="hostCreds" authTargetType="host"
                    defaultCredentialSet="HostCredsNormal"/>
      </credentials>
      <stepset ID="main" type="serial">
        <step ID="s1" command="putFile" >
          <paramList>
            <param name=sourceType>inline</param>
            <param name="destFile">%app-home%/%job_execution_id%.env</param>
            <param name="targetName">%job_target_names%[1]</param>
            <param name="targetType">%job_target_types%[1]</param>
            <param name=contents">
            <![CDATA[#!/bin/ksh
            export ENVVAR1=%param1% export ENVVAR2=%param2% export ENVVAR3=%param3%
            ]]>
            </param>
          </paramList>
        </step>
<step ID="s2" command="remoteOp" >
  <credList>
      <cred usage="defaultHostCred" reference="hostCreds"/>
  </credList>
    <paramList>
      <param name="remoteCommand">myscript</param>
      <param name="targetName">%job_target_names%[2]</param>
      <param name="targetType">%job_target_types%[2]</param>
      <param name="args">%app-home%/%job_execution_id%.env</param>
      <param name="successStatus">3</param>
      <param name="failureStatus">73</param>
    </paramList>
  </step>
<step ID="s3" command="remoteOp" >
  <credList>
      <cred usage="defaultHostCred" reference="hostCreds"/>
  </credList>

  <paramList>
    <param name="remoteCommand">rm</param>
    <param name="targetName">%job_target_names%[2]</param>
    <param name="targetType">%job_target_types%[2]</param>
    <param name="args">-f, %app-home%/%job_execution_id%.env</param>
    <param name="successStatus">0</param>
  </paramList>
</step>
</stepset>
<displayInfo useDefaultCreateUI="true"/>
</jobtype>
```

**Example 5**

The following example illustrates the use of the repSQL command to execute SQL
statements and anonymous PL/SQL blocks against the Management Repository.
The job type specification below calls a SQL statement in the first step S1, and
a PL/SQL procedure in the second step. Note the use of the variables %job_id%
and %job_name%, which are special job-system placeholders. Other job parameters
can be similarly escaped as well. Also note the use of bind parameters in the SQL
queries. The parameters sqlinparam[n] can be used to specify bind parameters.
There must be one parameter of the form sqlinparam[n] for each bind parameter.

Bind parameters must be used as far as possible to make optimum use of database resources.

**Example: Defining a Job Type That Executes SQL Statements and PL/SQL Procedures**

```
<jobtype name="repSQLJob" editable="true" version="1.0">
  <stepset ID="main" type="serial">
    <step ID="s1" command="repSQL" >
      <paramList>
        <param name="sql">update mytable set status='executed' where
         name=?</param>
        <param name="sqlinparam1">%job_name%</param>
      </paramList>
    </step>
  <step ID="s2" command="repSQL" >
<paramList>
  <param name="sql">begin mypackage.job_done(?,?,?); end;</param>
  <param name="sqlinparam1">%job_id%</param>
  <param name="sqlinparam2">3</param><param name="sqlinparam3">mgmt_rep</param>
</paramList>
</step>
</stepset>
<displayInfo useDefaultCreateUI="true"/>
</stepset>
</jobtype>
```

**Example 6**

This example illustrates the use of the switch stepset. The main stepset of this job is a switch stepset where switchVarName is a job parameter called stepType. The possible values (switchCaseVal) that this parameter can have are "simpleStep", "parallel", and "OSJob", which will end up selecting, respectively, the step SWITCHSIMPLESTEP, the parallel stepset SWITCHPARALLELSTEP, or the nested job J1.

```
<jobType version="1.0" name="SwitchSetJob" editable="true">
            <stepset ID="main" type="switch" switchVarName="stepType" >
  <credentials>
    <credential usage="hostCreds" authTargetType="host"
                defaultCredentialSet="HostCredsNormal"/>
  </credentials>

<step ID="SWITCHSIMPLESTEP" switchCaseVal="simpleStep" command="remoteOp">

  <credList>
    <cred usage="defaultHostCred" reference="hostCreds"/>
  </credList><paramList>
    <param name="remoteCommand">%command%</param>
    <param name="args">%args%</param>
    <param name="targetName">%job_target_names%[1]</param>
    <param name="targetType">%job_target_types%[1]</param>
  </paramList>
</step>
<stepset ID="SWITCHPARALLELSTEP" type="parallel" switchCaseVal="parallelStep">
  <step ID="P11" command="remoteOp" >
    <credList>
      <cred usage="defaultHostCred" reference="hostCreds"/>
    </credList>
    <paramList>
      <param name="remoteCommand">%command%</param>
```

```
              <param name="args">%args%</param>
              <param name="targetName">%job_target_names%[1]</param>
              <param name="targetType">%job_target_types%[1]</param>
          </paramList>
      </step>
      <step ID="P12" command="remoteOp" >
        <credList>
          <cred usage="defaultHostCred" reference="hostCreds"/>
        </credList>
      <paramList>
        <param name="remoteCommand">%command%</param>
        <param name="args">%args%</param>
        <param name="targetName">%job_target_names%[1]</param>
        <param name="targetType">%job_target_types%[1]</param>
      </paramList>
</step>
</stepset>
<job ID="J1" type="OSCommandSerial" switchCaseVal="OSJob" >
  <paramList>
    <param name="command">%command%</param>
    <param name="args">%args%</param>
  </paramList>
  <targetList>
    <target name="%job_target_names%[1]" type="%job_target_types%[1]" />
  </targetList>
</job>
</stepset>
<displayInfo useDefaultCreateUI="true"/>
</jobType>
```

**Example 7**

This example shows the use of the <securityInfo> tag to ensure that only users that have CLONE FROM privilege over the first target and MAINTAIN privilege over the second target are able to submit jobs of the following type:

```
<jobType name="Clone" editable="true" version="1.0" >
  <securityInfo>
    <privilege name="CREATE TARGET" type="system" />
    <privilege name="CLONE FROM" type="target" evaluateAtSubmission="false" >
      <target name="%job_target_names%[1]" type="%job_target_types%[1]" />
    </privilege>
    <privilege name="MAINTAIN" type="target" evaluateAtSubmission="false">
      <target name="%job_target_names%[2]" type="%job_target_types%[2]" />
    </privilege>
  </securityInfo>
  <!-- An optional <paramInfo> section will follow here, followed by the stepset
definition of the job
  -->
  <paramInfo>
  ....
  </paramInfo>
  <stepset ...>
  .......
  </stepset>
<displayInfo useDefaultCreateUI="true"/>
</jobType>
```

**Example 8**

The following shows an example of a scenario where credentials are passed to a nested job in the job type specification:

```
<jobType version="1.0" name="SampleJobType001" singleTarget="true"
editable="true"
 defaultTargetType="host" targetTypes="all">
 <credentials>
  <credential usage="osCreds" authTargetType="host"
   defaultCredentialSet="HostCredsNormal" credentialTypes="HostCreds">
    <displayName nlsid="LABEL_NAME">OS Credentials</displayName>
    <description nlsid="LABEL_DESC">Please enter credentials.</description>
  </credential>
 </credentials>
 <stepset ID="main" type="serial">
  <step ID="Step" command="remoteOp">
   <credList>
    <cred usage="defaultHostCred" reference="osCreds" />
   </credList>
   <paramList>
    <param name="targetName">%job_target_names%[1]</param>
    <param name="targetType">%job_target_types%[1]</param>
    <param name="remoteCommand">/bin/sleep</param>
    <param name="args">1</param>
   </paramList>
  </step>
 <job ID="Nested_Job" type="OSCommand">
  <credList>
   <cred usage="defaultHostCred" reference="osCreds" />
  </credList>
  <targetList allTargets="true" />
  <paramList>
    <param name="command">/bin/sleep</param>
    <param name="args">1</param>
   </paramList>
  </job>
 </stepset>
</jobType>
```

# About Performance Issues

This section provides a brief discussion on issues to consider when designing your job type. These issues might impact the performance of your job type as well as the overall job system.

# Using Parameter Sources

The following issues are important in relation to the use of parameter sources:

- Parameter sources are a convenient way to obtain required parameters from known sources, such as the Management Repository or the credentials table. The parameter sources must be used only for quick queries that fetch information stored somewhere else.

- Parameter sources that are evaluated at job execution time will, in general, effect the throughput of the job dispatcher and must be used with care. In some cases, the fetching of parameters at execution time might be unavoidable and if you do not care whether the parameters are fetched at execution time or submission time, set evaluateAtSubmission to false.

- When executing SQL queries to obtain parameters (using the SQL parameter source), the usual performance improvement guidelines apply. These include using indexes only where necessary and avoiding the joining of large tables.

# Adding a Job Type to Enterprise Manager

To package a new job type with a metadata plug-in, you must adhere to the following implementation guidelines:

New job types packaged with a metadata plug-in have two new files:

- Job type definition XML file: Used by the job system during plug-in deployment to define your new job type. There is one XML file for each job type.

- Job type script file: Installed on selected Management Agents during plug-in deployment. A single script might be shared amongst different jobs.

The following two properties must be set to "true" in the first line of the job type definition XML file:

- agentBound

- singleTarget

Here is an example:

```
<jobType version="1.0" name="PotatoUpDown" singleTarget="true" agentBound="true"
targetTypes="potatoserver_os">
```

Because the use of Java for a new job type is not supported for job types packaged with a plug-in, new job types are agentBound and perform their work through a script delivered to the Management Agent (the job type script file). The job type definition XML file contains a reference to the job type script file and executes it on the Management Agent whenever the job is run from the Enterprise Manager console.

## Adding a Job Type to an Oracle Plug-in Archive (OPAR)

After you have created the job type definition XML file and modified the target type definition file, add your files to an Oracle Plug-in Archive (OPAR) just as you would any other target type. See Validating, Packaging, and Deploying the Plug-in for more information.

# 9

# Defining a Management User Interface

Enterprise Manager can be extended to support the management of new domains through the introduction of discovery, monitoring, and automation. While the Enterprise Manager framework provides a powerful set of features related to these management capabilities, most plug-in developers need to expose management capabilities in a way that is appropriate to their domain. The Metadata Plug-in Custom User Interface (MPCUI) features of Enterprise Manager provide you with this capability.

This chapter contains the following sections:

- Introduction to Defining a Management User Interface
- MPCUI Concepts
- Creating a Custom UI for a Plug-in
- Creating the MPCUI Metadata File
- Defining Metadata
- Defining the MPCUI Application
- Packaging the MPCUI Implementation With the Plug-in
- Defining System Home Pages
- Defining Navigation
- Accessing Enterprise Manager Data
- Performing Task Automation
- Storing Session State
- Defining Page Layout Components
- Including Packaged Regions
- Defining Charts
- Defining Tables
- Defining Dialogs
- Defining Trains
- Defining Information Item and Information Displays (Label-Value Pairs)
- Using Built-in Renderers
- Defining Links
- Including Enterprise Manager Images
- Displaying a Processing Cursor
- Defining Icons for Target Types
- Displaying the Target Navigator
- Defining a UI for Guided Discovery

# Introduction to Defining a Management User Interface

As a plug-in developer, you are responsible for the following steps for defining a custom user interface for managing your target types:

> **Note:**
>
> In addition to this document, the Extensibility Development Kit (EDK) includes a complete sample implementation that should be used as a guide during this process.

1. Decide on the model for your target including:
   - Associations with other targets
   - Performance metrics and configuration data
   - Subcomponents of the target
   - Administrative tasks and operations
2. Familiarize yourself with the capabilities provided by the MPCUI library, such as:
   - UI components that are available (pages, charts, and so on)
   - Services that are available (metric data, SQL query, associations, task execution, and so on)
   - Sample implementations and how they are constructed
3. Design the UI based on:
   a. Data and tasks that are important
   b. Capabilities provided by MPCUI and JavaScript Extension Toolkit (JET)

   This can involve drawing the pages and describing their content, and reviewing the page with domain experts to ensure they expose the appropriate management capabilities.
4. Create the target metadata for the items in your design (see step 1). This metadata is necessary to implement your UI later. For more information about target metadata, see the relevant chapters within this guide.
5. Develop the SQL queries required to retrieve configuration data that will be displayed in the UI. Typically, these queries reference the configuration `CM$` views.

For more information about configuration data, see Collecting Target Configuration Data.

6. Identify and define the activities that make up your UI, such as pages, wizards, and dialogs. The Integration metadata defines these activities.

   For more information, see Defining Integration Metadata.

7. Implement your custom user interface using MPCUI and JET.

   For more information, see Oracle JavaScript Extension Toolkit (Oracle JET).

## HTML/JavaScript (JS) Implementation

You are responsible for the following steps:

1. Obtain a copy of NetBeans or comparable IDE.

   For more information, see Development Environment Options.

   > **Note:**
   >
   > NetBeans is not required, but the examples will be available in a format (nbm) which ties them into NetBeans project creation flow. Example source also available in a zip file for use with an alternate IDE.

2. Create a project to hold the source code for your custom UI. You can use the sample project included in the EDK as a template, or you can use the MPCUI Starter project which has the MPCUI JS library and all of the JET libraries but none of the implemented pages from the samples.

   For more information, see Developing MPCUI in NetBeans.

3. Create the MPCUI metadata file. This defines the set of activities included in the custom UI.

   This file includes:

   • SQL statements used by your custom UI

   • Menu items you want to include to support navigation to different pages defined in your UI

   • Reference to the JS library you built for your custom UI.

   • Activity/Dialog/Train definitions used by the MPCUI to navigate through your UI.

   For more information, see Creating the MPCUI Metadata File or Defining the Application Activities.

4. Develop each activity (such as page or dialog). Typically, each page includes a page class (an HTML file) and a controller class (written in JavaScript extending the ActivityController class).

   For more information, see Defining Pages, Defining Dialogs, and Defining Trains and Train Pages.

5. Build your JS library and test your custom UI from NetBeans.

> **Note:**
>
> You must deploy at least one version of your plug-in before building and testing. The deployed plug-in must include the target metadata (such as metrics and configuration data). However, the plug-in does not have to include your MPCUI metadata for testing.

6. Modify your plug-in to include the MPCUI metadata file and the JS library you built.

   Place these files in the oms/metadata/mpcui directory of the plug-in staging area.

   For more information, see Packaging the MPCUI Implementation With the Plug-in.

7. Test your custom UI by accessing a target home page from the Enterprise Manager console.

   This loads your custom UI in the context of the Enterprise Manager application and displays the Enterprise Manager application and target menus.

In addition to this document, additional resources for developing with MPCUI/JET components are provided:

- The API reference: This is located in your partner EDK directory under doc/sdk_api_ref.html

- The HostSample example plug-in: The sample plug-in provided by Oracle provides examples of many MPCUI features. It is located in the EDK under samples/plugins/HostSample

You may also include any of the base HTML/JS/JET components (such as Button, Label, and so on). Oracle develops the JET library, and you can find the documentation for the JET library online at the following link:

http://www.oracle.com/technetwork/developer-tools/jet/overview/index.html

## Assumptions and Prerequisites

This chapter assumes you are familiar with the following:

- Plug-in development overview, including how to package a plug-in and its XML files

- HTML and JS technologies

- JET and its dependent libraries (jQuery, knockout, require)

# MPCUI Concepts

There are several important concepts that should be understood when using the MPCUI framework. These concepts are defined briefly in this section and discussed in more detail in the subsequent sections.

## MPCUI Metadata File

The MPCUI metadata file contains the bootstrap for your UI, which is used to define the set of pages, dialogs, and trains that are included in the UI. The MPCUI framework uses this information to drive the UI including managing navigation between UI elements.

## Activity

Top-level UI elements in the MPCUI are referred to generally as activities. Activities include pages, dialogs, trains and train pages, URLs, and jobs.

## Page

This is a construct that is provided by the MPCUI framework to simplify the construction of the UI and make it fit more naturally into the larger Enterprise Manager console. Typically, this will refer to an HTML file you create.

The MPCUI framework manages pages within the application, providing simple navigation between pages and integrating them into the browser history and the Enterprise Manager menu system.

## Services

The MPCUI framework provides a series of services that can be used to retrieve data from the Management Server or to process actions (jobs or remote operations).

## Data Services

The Data Services provided by MPCUI include data services to retrieve metric data, associations, target properties and so on. It includes a `SQLDataService` that can be used to run named SQL statements within the plug-in.

## Operation Services

MPCUI includes a Job service and RemoteOp service that can be used to perform administrative actions against the targets managed by the plug-in code.

- The Job service requires the inclusion of job type definitions in the plug-in
- The RemoteOp service requires the registration of scripts with the plug-in framework

## Asynchronous Service Request Handling

The MPCUI framework handles network requests asynchronously. This requires the use of a result handler pattern where a request is made to the server and as part of the request, a handler (or callback) is registered with the request. Upon completion of the request (or if a fault occurs), the handler is called and passed the result.

## URL

MPCUI provides a number of different capabilities related to the generation of URLs and the ability to embed links to:

- Other Enterprise Manager pages
- Other pages within the MPCUI application
- External pages

# Creating a Custom UI for a Plug-in

You can create a custom plug-in UI via HTML/JS.

## HTML/JS Implementation

To provide a custom UI with a plug-in, you must provide HTML pages for display and JS for any programmatic conroller logic. The capabilities for what is possible with this implementation is set by JET, which provides a variety of implementations and standards to make developing a page or suite of pages easier.

While one of the goals of the MPCUI framework is to provide a simplified layer of abstraction over the HTML/JS/JET framework with which it is implemented, you must become familiar with the HTML/JS, the JET framework, and the JS libraries upon which JET depends.

- HTML
- JavaScript
- JS Library File

## HTML

The HTML page will determine the structure of your page, its layout, and what appears to the plug-in user. Much of what you do using MPCUI can be accomplished in HTML.

## JavaScript

For cases that require more complex handling of data or events, you might have to develop part of the UI using JavaScript. All controller logic is implemented using JS.

## JS Library File

When creating a custom UI, the final product will be delivered as a JS library file. All HTML and JS created to run your plug-in UI are combined into a single JS library file. This is what is packaged as a part of your plug-in. At runtime, the Enterprise Manager wrapper page dynamically generates the required references to your JS library and displays it for the plug-in user.

> **Note:**
>
> You can provide a minified version of your JS library for normal runtime and also a debug version of the JS library, the use of which can be triggered with the setting of a query parameter on the page.

# Creating the MPCUI Metadata File

Each plug-in that includes MPCUI must include an MPCUI metadata file.

The metadata file:

- Defines SQL queries required by the MPCUI

- Defines the menu items required by the MPCUI

- Contains UI metadata for the target UI

- Contains UI metadata for discovery

- Specifies target icons, target navigator, and system home page options

For more information about the syntax for this file, see the XSD file located in the Extensibility Development Kit (EDK) specifications.

The following examples provide a summary of the metadata-based UI MPCUI metadata file and a summary of the UI metadata file

**Example: MPCUI Metadata File**

```
  <CustomUI target_type="demo_hostsample"
xmlns="http://www.oracle.com/EnterpriseGridControl/MpCui">

   <!-- SqlStatements defines the individual SQL statements that are used by
      the MPCUI code.  Each statement is identified by a unique name and
      can only be referenced by that name from the MPCUI code itself -->
   <SqlStatements>
     <Sql name="INSTANCE_INFO">
         select * from...
     </Sql>
   </SqlStatements>

 <UIMetadata>
   <Integration>
   .....
   </Integration>

 </UIMetadata>

   <!-- MenuMetadata defines the set of menu items that should appear in the
        target menu on the homepage and specifies which of the MPCUI pages
        should be accessed from that menu item -->
   <MenuMetadata>
     <menu label="Host Sample">
       <menuItem>
         <command .. />
       </menuItem>
     </menu>
   </MenuMetadata>

   <EmuiConfig>
     <context-pane-visible>true</context-pane-visible>
     <large-icon>dhs_large.png</large-icon>
     <small-icon>dhs_small.png</small-icon>
     <use-framework-homepage>true</use-framework-homepage>
   </EmuiConfig>

 </CustomUI>
```

Everything within the <Integration> tag will define the metadata which runs your custom UI.

# Overview of MPCUI Metadata Elements

Table 9-1 describes the key elements that define the metadata.

**Table 9-1    Key Elements Used to Define Discovery Metadata**

| Element | Description |
|---|---|
| SqlStatements | The `SqlStatements` element contains the SQL statements that enable you to access information stored in the Management Repository. For more information about these SQL statements, see Packaged SQL and the Query Service. |
| UIMetadata | The `UIMetadata` element is the top-level container for the integration and page (activity) definitions described by that metadata:<br><br>`<UIMetadata>`<br><br>`  <!-- The meta-data only definition must include an Integration element`<br>`  which defines the set of activities (pages, dialogs, etc.) that make up`<br>`  the application -->`<br>`  <Integration>`<br>`    ...`<br>`  </Integration>`<br><br>`</UIMetadata>` |
| Integration | The `Integration` element defines the integration metadata used to specify the set of pages and to define task flows between these pages (if required). For information about integration metadata, see Defining Integration Metadata. |
| MenuMetadata | The `MenuMetadata` element includes the `menuItem` elements that define navigation to activities defined in the MPCUI metadata. For more information about the `MenuMetadata` element, see Defining Navigation. |
| EmuiConfig | The `EmuiConfig` element includes elements to define the following<br>• Target navigator (`context-pane-visible`)<br>  For more information, see Displaying the Target Navigator.<br>• Icons to represent target types in the Cloud Control console (`large-icon`, `small-icon`)<br>  For more information, see Defining Icons for Target Types.<br>• System home page (`use-framework-homepage`)<br>  For more information, see Defining System Home Pages. |

# Defining Metadata

For a complete example of an MPCUI metadata implementation, see the Demo Sample implementation (data/metadata/stage/demo_hostsample_uimd.xml) provided with the Extensibility Development Kit (EDK).

# Defining Integration Metadata

Use the integration metadata to specify the set of pages and to define task flows between these pages (if required).

**Example: Integration Metadata**

```
<Integration>

    <!--
        The mpcuiLibVersion determines what UI technology stack your code
        runs against.  The mpcuiLibVersion is backed by a specific version
        of MPCUI code, but it is also backed by a specific version of JET
        and the libraries that support JET (knockout, jquery, etc.)

          Setting this version guarantees that your code will perform correctly
          against this version of MPCUI and the version of JET which backs it.
          This defaults to MPCUI version 13.2.0.0.0 (backed by JET 2.3.0).
        -->


  <mp:Integration mpcuiLibVersion="13.2.0.0.0"
      xmlns:mp="http://www.oracle.com/EnterpriseGridControl/MpCuiIntegration"
  >
        <!--
            The sourceContext is the deploy time settings which help the MPCUI
identify
            where certain aspects of your UI are located so that they may be
installed to
            the Repository.

            These settings are simply a utility to keep the notation in the rest
of the
            file more brief and closer to what was specified in the integration
file from
          the Flex implementation.
        -->
    <mp:sourceContext>
      <!--
            The jsRoot is used as a prefix to any of the jsLibraries or
controller
            classes you may specify. It is only used for controller classes in
the
            case of an exploded deploy (where each file goes in separately).
This
            is not recommended for performance reasons.  It is much more
efficient
            to package all of your code, JS and HTML, as a library and deploy a
single
            file.
        -->
        <mp:jsRoot path="js"/>
          <!--
                The viewRoot is used as a prefix to any of the activities
class files
                (HTML, not controllers).  Only used if broken out files are
used instead
            of a JS library for delivering a UI.  It is much more preferable
and
                performant to specify a library, but this is an option as well.
```

```
            eg:  mp:viewRoot path="ui/view"
         -->
         <!-- The bundleRoot is used as the prefix to any path specified for
a
            resourceBundle -->
        <mp:bundleRoot path="rsc"/>

          <!-- The cssRoot is used as the prefix to any path specified for a
cssFile
             eg:  mp:cssRoot path="ui/view/css"
         -->
        <mp:cssRoot path="css/dhs"/>
      </mp:sourceContext>

     <!--
          Any css files deployed with the plug-in will be loaded in the
index.html
          file.
     -->
     <mp:cssFiles>
         <mp:cssFile id="myCss" path="dhs.css" version="13.1.0.1.0"/>
         </mp:cssFiles>

     <!--
          Any JS libraries used by the UI would be declared here.  This
includes any
          libraries the UI may depend upon outside of the ones already
required by
          the MPCUI and JET (so if you used a 3rd party library).  It also
includes
          the UI of the plug-in if it is packaged as a library.

          The entries here are used to dynamically generate the main.js file
used by
          the UI at runtime.  So any jsPath's listed here will be mapped to
that
          library in the paths at the top of the main.js file:

            requirejs.config({
                paths:
                {
                'knockout': 'libs/knockout/knockout-3.4.0',
                'jquery': 'libs/jquery/jquery-2.1.3.min',
                <jsPath.path>:<jsLibrary>
                ...

          Any jsShims listed here will be added as so:
             <jsShim.name>:
             {
               exports: '<jsShim.exports>',
               deps: ['<jsShim.deps[0]', '<jsShim.deps[1]',...]
             }

          jsShim.deps is a simple comma-delimited list of strings, which is
parsed
          to produce the output above.

              And any jsModules listed here will be added to the require
clause which
          instantiates the MPCUI application at the end of the main.js
file.
```

```
                There will be a default list of modules specified, but if you
need any
                additional ones you will have to list them here. They aren't all
added
                by default for runtime performance concerns.

                    require([
                    'ojs/ojcore',
                    'knockout',
                    'jquery',
                    'emx/intg/MpAppLoader',
                    'signals',
                    'ojs/ojmodel',
                    'ojs/ojknockout',
                    <jsModule.module>,
                    ...

          -->
     <mp:jsLibraries>
          <!--
              When the main.js file is generated, each activity controller is put
in the
              paths property mapping either to the jsLibrary marked as the
default or
              to the library it is specifically noted for (jsPath.activityId)
          -->
        <mp:jsLibrary id="pluginLib" path="libs/dhs/demo_hostsample-min.js"
                      debugPath="libs/dhs/demo_hostsample-debug.js"
                      version="13.2.0.0.0" isDefault="true">
          <mp:jsModule module="ojs/ojmodel"></mp:jsModule>
          <mp:jsModule module="ojs/ojknockout"></mp:jsModule>
          <mp:jsModule module="ojs/ojknockout-model"></mp:jsModule>
          <mp:jsModule module="ojs/ojcomponents"></mp:jsModule>
          <mp:jsModule module="ojs/ojarraytabledatasource"></mp:jsModule>
          <mp:jsModule module="ojs/ojdatetimepicker"></mp:jsModule>
          <mp:jsModule module="ojs/ojtable"></mp:jsModule>
          <mp:jsModule module="ojs/ojdatagrid"></mp:jsModule>
          <mp:jsModule module="ojs/ojchart"></mp:jsModule>
          <mp:jsModule module="ojs/ojgauge"></mp:jsModule>
          <mp:jsModule module="ojs/ojlegend"></mp:jsModule>
          <mp:jsModule module="ojs/ojselectcombobox"></mp:jsModule>
          <mp:jsModule module="ojs/ojsunburst"></mp:jsModule>
          <mp:jsModule module="ojs/ojthematicmap"></mp:jsModule>
          <mp:jsModule module="ojs/ojtreemap"></mp:jsModule>
          <mp:jsModule module="ojs/ojvalidation"></mp:jsModule>
          <mp:jsModule module="ojs/ojslider"></mp:jsModule>
          <mp:jsModule module="ojs/ojpagingcontrol"></mp:jsModule>
        </mp:jsLibrary>
        <!--
            There can be only 1 default library.  Any classes that aren't
attached to
            another library will be attached to the default library.

            If not the default, you can associate an activity with the library
with:
              mp:jsPath id="activityId"

            and the appropriate path will be added for that activity or the
explicit
            path can be set:
              mp:jsPath path="dhs/MyController"
```

```
         Shims may also be specified:
           mp:jsShim name="myshim" exports="myshim" deps="jquery, ojs/ojcore"
        -->
    </mp:jsLibraries>


    <!--
         Resource bundles used by this application.  They can be declared
         for the entire application or on a page-by-page basis as desired
    -->
    <mp:resourceBundles>
       <mp:MpBundle name="demoUiMsg" path="oracle.samples.xohs.rsc"
isDefault="true"/>
       <mp:MpBundle name="demoJobMsg" path="oracle.samples.xohs.rsc"/>
    </mp:resourceBundles>


    <mp:activities>
      <!--
         Each page definition must have an id that is *different* than the page
class,
         and must have a pageClass that references the HTML file that lays out
the
         page.  If the page includes a custom controller to do event handling
then the
         pageControllerClass is set to point to the JavaScript class that
defines
         the PageController extensions for this page.  Finally, one of the
pages in
         the list should include the "isDefaultPage='true'" designation to
indicate
         the 1st page to be loaded.
      -->

      <!-- Pages -->
      <mp:PageActivityDef id='homePg' label='Home' pageClass='dhs/HomePage'
                          pageControllerClass='dhs/HomePageController'
                          isDefaultPage="true" />

      <mp:PageActivityDef id='perfPg' label='Performance' pageClass='dhs/
PerfPage'
                          pageControllerClass='dhs/PerfPageController' />

      <mp:PageActivityDef id='adminPg' label='Administration'
                          pageClass='dhs/CredentialsPage'
                          pageControllerClass='dhs/
CredentialsPageController'/>

      <mp:PageActivityDef id='filesystemsPg' label='Filesystems'
                          pageClass='dhs/FilesystemsPage'
                          pageControllerClass='dhs/
FilesystemsPageController'/>

      <mp:PageActivityDef id='processesPg' label='Processes'
                          pageClass='dhs/ProcessesPage'
                          pageControllerClass='dhs/
ProcessesPageController' />

      <mp:PageActivityDef id='collectionsPg' label='Collections'
                          pageClass='dhs/CollectItemPage'
                          pageControllerClass='dhs/
CollectItemPageController' />
```

```
        <mp:PageActivityDef id='homeExtModelPg' label='Home (Ext Model)'
                            pageClass='dhs/HomePageExtModel'
                            pageControllerClass='dhs/HomePageModelController' />


        <!-- Trains -->
        <mp:TrainActivityDef id='addNewUserEmbeddedTrain' label='Add New User'>
           <mp:stepActivities>
              <mp:TrainStepActivityDef id='anuStep1' label='User Info'
                            pageClass='dhs/user/UserInfo'
                            pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
              <mp:TrainStepActivityDef id='anuStep3' label='Credentials'
                            pageClass='dhs/user/Credentials'
                            pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
              <mp:TrainStepActivityDef id='anuStep2' label='Expiry'
                            pageClass='dhs/user/Expiry'
                            pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
              <mp:TrainStepActivityDef id='anuStep4' label='Schedule'
                            pageClass='dhs/user/Schedule'
                            pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
              <mp:TrainStepActivityDef id='anuStep5' label='Notifications'
                          pageClass='dhs/user/Notifications'
                          pageControllerClass='dhs/user/
NotificationsTrainStepController'/>
              <mp:TrainStepActivityDef id='anuStep6' label='Confirmation'
                          pageClass='dhs/user/Confirm'
                          pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
           </mp:stepActivities>
        </mp:TrainActivityDef>
        <!-- Add new filesystem train activity definition -->
        <mp:TrainActivityDef id='addNewFSCreateTrain' label='Add New Filesystem'
                    trainControllerClass='dhs/filesystem/
AddNewFilesystemTrainController'>
           <mp:stepActivities>
             <mp:TrainStepActivityDef id='anfStep1' label='Filesystem Info'
                       pageClass='dhs/filesystem/FilesystemInfo'
                       pageControllerClass='dhs/filesystem/
AddNewFilesystemStepController'/>
             <mp:TrainStepActivityDef id='anfStep2' label='Credentials'
                       pageClass='dhs/filesystem/Credentials'
                       pageControllerClass='dhs/filesystem/
AddNewFilesystemStepController'/>
             <mp:TrainStepActivityDef id='anfStep3' label='Confirmation'
                       pageClass='dhs/filesystem/Confirm'
                       pageControllerClass='dhs/filesystem/
AddNewFilesystemStepController'/>
           </mp:stepActivities>
        </mp:TrainActivityDef>


        <!--
            Dialog activities are defined similar to pages and may also define
parameters
            that should be set when accessing the dialog through invokeActivity
calls.
            If these parameters are specified, then an input object can be
provided and
```

```
        the framework will attempt to retrieve the values for these properties
from
        the input.  This is often done by calling the "bean()" method to
construct
        the context but any object that includes the required properties may be
        passed
    -->
    <mp:DialogActivityDef id='metricHistExtModel' label='Metric History'
                          dialogClass='dhs/MetricHistoryExtModelDlg'
                          dialogControllerClass='dhs/
MetricHistoryDlgModelController'>
       <mp:inputParams>
         <mp:InputParam name='targetName'/>
         <mp:InputParam name='targetType'/>
         <mp:InputParam name='metricName'/>
         <mp:InputParam name='metricColumns'/>
         <mp:InputParam name='timePeriod'/>
         <mp:InputParam
name='title'/>

       </mp:inputParams>
    </mp:DialogActivityDef>

    <mp:DialogActivityDef id='metricHistory' label='Metric History'
                          dialogClass='dhs/MetricHistoryDialog'
                          dialogControllerClass='dhs/
MetricHistoryDialogController'>
       <mp:inputParams>
         <mp:InputParam name='targetName'/>
         <mp:InputParam name='targetType'/>
         <mp:InputParam name='metricName'/>
         <mp:InputParam name='metricColumns'/>
         <mp:InputParam name='timePeriod'/>
         <mp:InputParam
name='title'/>

       </mp:inputParams>
    </mp:DialogActivityDef>


    <mp:DialogActivityDef id='searchProcDialog' label='Search Processes'
                          dialogClass='dhs/SearchProcessDialog'
                          dialogControllerClass='dhs/
SearchProcessDialogController'>
       <mp:inputParams>
         <mp:InputParam name="searchState" required="false" />
       </mp:inputParams>
    </mp:DialogActivityDef>

    <mp:DialogActivityDef id='credentialsDialog' label='Credentials'
                          dialogClass='dhs/CredentialsDialog' >
       <mp:inputParams>
         <mp:InputParam name="credState" required="false" />
       </mp:inputParams>
    </mp:DialogActivityDef>

    <mp:DialogActivityDef id='fsCreateTrainDialog' label='Add New Filesystem'
              dialogClass='dhs/filesystem/FSCreateTrainDialog'
              dialogControllerClass="dhs/filesystem/
FSCreateTrainDialogController" />
    <mp:DialogActivityDef id='rpmInfoDialog' label='RPM Info'
```

```
                                    dialogClass='dhs/RPMInfoDialog'
                                    dialogControllerClass="dhs/
RPMInfoDialogController" />
    </mp:activities>
  </mp:Integration>
<Integration>
```

# Defining Navigation

References to the page implementations are defined in the metadata in the previous section, but the pages will not appear in the target instance specific menu unless specifically added to that section of the XML file:

Defining a menu item in the metadata that can be used to access a page

The `MenuMetadata` item includes the `menuItem` elements that define navigation to activities defined in the MPCUI metadata. For example, if the metadata includes the following page definition:

```
 <mp:PageActivityDef id='processesPg' label='Processes' …/>
```

Specify a `menuItem` in the `MenuMetadata` element to allow navigation to the previous page:

```
  <menuItem>
   <command id="processesPg" label="Processes"
              class="oracle.sysman.emSDK.pagemodel.menu.EMNavigationMenuCommand
              partialSubmit="true" >
      <property name="actionOutcome" value="goto_core-mpcustom-nav" /><property
name="paramsMap"><mapEntry name="pageid" value="processesPg" />
      </property>
    </command>
 </menuItem>
```

The key properties in the `menuItem` element are:

* `label` within the command element.

    `label` specifies the label that appears in the target menu on the home page. In the example given, a menu item "Processes" would be included.

* the value specified for the `actionOutcome` property.

    `actionOutcome` specifies the view ID for the page containing the SWF file.

# Defining the MPCUI Application

The basis for the custom UI built using the MPCUI framework requires the construction of an HTML/JS-based application. To simplify this process, the framework provides a series of base classes and structures.

The Integration Metadata in the MPCUI metadata is the core piece of the MPCUI application. It is used by the MRS to deploy the custom UI correctly. It is used by the runtime MPCUI wrapper page to define the relationships between pages. It is used to populate the menu items for the MPCUI page. (It is also used for these last 2 to achieve the same things in the standalone console when running out of the IDE).

## Defining the Application Activities

The MPCUI framework interacts with the integration metadata to understand the structure of the application, allowing the framework to be the primary driver behind the display of and navigation between the UI elements that make up the application.

The application activities are registered solely in the MPCUI integration metadata.

**Example: Application Activities**

```
<Integration>
  <mp:Integration mpcuiLibVersion="13.2.0.0.0"
      xmlns:mp="http://www.oracle.com/EnterpriseGridControl/MpCuiIntegration"
  >

...

      <!-- The integration class defines the pages, dialogs
              and trains included in the application -->
    <mp:activities>
      <!--
          Each page definition must have an id that is *different* than the page
class,
          and must have a pageClass that references the HTML file that lays out
the
          page.  If the page includes a custom controller to do event handling
then the
          pageControllerClass is set to point to the JavaScript class that
defines
          the PageController extensions for this page.  Finally, one of the
pages in
          the list should include the "isDefaultPage='true'" designation to
indicate
          the 1st page to be loaded.
      -->

      <!-- Pages -->
      <mp:PageActivityDef id='homePg' label='Home' pageClass='dhs/HomePage'
                          pageControllerClass='dhs/HomePageController'
                          isDefaultPage="true" />

      <mp:PageActivityDef id='perfPg' label='Performance' pageClass='dhs/
PerfPage'
                          pageControllerClass='dhs/PerfPageController' />


    </mp:activities>
</mp:Integration>
```

## Defining Pages

Each page must be registered with the MPCUI framework through the Integration metadata by adding a `PageActivityDef`. The `PageActivityDef` is defined by:

- Page

  The page is the concrete implementation of the page, that is its layout and contents and is an HTML file.

- Page controller

    The page controller is a class that extends the `ActivityController` base class and encapsulates the set of handlers that support interacting with the Enterprise Manager services layer to obtain data and bind it to the UI components and respond to events issued by the UI on behalf of the end-user (e.g.button presses or link clicks)

Each application must include at least one page (one page activity) and you must identify one of the page activities as the default page.

> **✎ Note:**
>
> The default page is displayed by the MPCUI framework as the home page for the selected target

## Page

The `Page` is the top-level UI element in the application. The framework provides integration of pages into the Enterprise Manager console by::

- integrating pages with the Enterprise Manager menu system

- performing updates of the browser history so that pages can be bookmarked

- providing simple navigation between pages

Implement pages in HTML. The tag language that is used to describe the page includes a mix of basic HTML, JET components, and MPCUI-provided components for layout and data display. The description of each component and example for its use are included in subsequent sections of this document.

For examples of the page class, see the HomePage.html and ProcessesPage.html files from the Demo HostSample in the EDK.

## Page Model

Components within the page display information obtained through the Enterprise Manager services layer, and typically are bound to this data through the page model. The page model is the set of data associated with the page. The framework manages the lifecycle of this data so that as pages are displayed, data is loaded. When pages are removed, the data is cleaned up.

Specify the data included in the page model by:

- using data service tags

- adding data directly to the page model in the result handlers for Enterprise Manager service requests

For additional information about describing the use of the service layer and how data is added to the model, see Performing Task Automation.

The page model is implemented through the KnockoutJS library, which makes model objects dynamic when bound in the HTML page. So references to items on the page model from components in the HTML will be dynamically updated when the model is changed in the controller. Oracle recommends that the Page code is limited to the

layout of the UI elements that make up the page. Delegate data binding and event handling to the controller. This ensures that the MPCUI framework can manage the lifecycle of each page and the data bound to it correctly.

## Page Controller

The page controller is a class that extends the PageController base class and includes the code that interacts with the Enterprise Manager services layer to obtain data and to process administrative actions. Furthermore, the controller contains the set of event handlers that are called in response to events issued from the Page components.

> **✎ Note:**
>
> A page controller is not necessary if all of the data displayed in the page can be specified through the `component` tags or the `DataService` tags and custom event handling is not necessary.
>
> For example, if a page is a container for a number of Chart components, then each component supports the specification of the metric to be displayed in the chart. The component interacts with the MPCUI framework to manage the life cycle of that data correctly.

For cases where a controller is necessary, the `init(page)` method is the location in the code where you can load data to be bound to the page UI elements. For examples for interacting with Enterprise Manager services and binding using the page model, see Performing Task Automation.

In addition to the `init` method, the controller includes methods that respond to events originating in the page. In cases where it is necessary to perform some processing in response to an event (for example, a button press), you can reference a method in the controller that will be called when that event occurs.

- Within the Page:

```
<mp-link id="showHistory" params="label: getString('SHOW_HISTORY'),
                    destination: cb(controller.showHistoryDialog,
                                    bean('title', 'Title', 'metricName',
                                        'CPUProcessorPerf',
'metricColumns',
                                        ['CPUIdle'], 'timePeriod',
'LAST_DAY'))">
  </mp-link>
```

- Within the Controller:

```
HomePageController.prototype.showHistoryDialog = function(context, event) {
      this.page.invokeActivity('metricHistory', context);
    };
```

In the page code, a reference to `controller` is all that is necessary to interact with code included in the page controller. The framework manages creating the controller class when the page is loaded and provides the ability to call through into the controller to take some action.

The framework simplifies the process for taking some actions by providing convenience methods that can be called directly from the Page without requiring

additional event handlers in the controller. For example, accessing another activity can be done in most cases without requiring additional controller code.

In the following example, clicking the link redirects the application to the `processesPg` activity.

```
<mp-link id="procLink" params="label: 'Show Process',
                        destination: invokeActivityCb('processesPg')">
</mp-link>
```

> **Note:**
>
> For more information, see the HomePageController.js and ProcessesPageController.js files from the Demo Sample.

## Defining Dialogs

Dialogs are popup windows that display on top of the application without navigating away from the current Page displayed. Dialogs are defined in HTML files and do not have separate controller classes (although they can).

```
<mp-dialog params="mpDialog : { height:360, width:450 }" >
    <mp-row>
      <mp-column>
            <div style="width:100%;height:35px">
                <select id="selMemChart" data-bind="ojComponent: {component:
'ojSelect',
                    options: model().timePeriodList, value:
model().selectTimePeriod, optionChange: cb(controller.changeChart),
                    rootAttributes: {style:'max-width:20em'} }">
                </select>
            </div>
            <mp-chart id="metDataCustBinding"
                 params="mpChart : { type: 'line',
                  dataSelection: 'multiple',
                  emptyText: 'No data',
                  legend: {rendered: false},
                  yAxis: {min: 0, max: 100},
                  styleDefaults: {colors: Colors.DEFAULT_COLORS},
                  animationOnDisplay: 'auto',
                  targetName: appModel.target.name,
                  targetType: appModel.target.type,
                  metricName: model().metricName,
                  metricColumns: model().metricColumns,
                  timePeriod: model().timePeriod}"
                 style="width:100%;height:calc(100% - 35px)" >
            </mp-chart>
      </mp-column>
    <mp-row>
</mp-dialog>
```

In the previous example, the dialog references model as the source of the properties it uses in the UI components.

Initialize the dialog model either:

- In a controller associated with the dialog

- By the MPCUI framework if the Dialog definition in the Integration class specifies input parameters

```
<mp:DialogActivityDef id='metricHistory' label='Metric History'
                          dialogClass='MetricHistoryDialog' >
    <mp:inputParams>
        <mp:InputParam name='targetName'/>
        <mp:InputParam name='targetType'/>
        <mp:InputParam name='metric'/>
        <mp:InputParam name='columns'/>
        <mp:InputParam name='period'/>
        <mp:InputParam name='title'/>
    </mp:inputParams>
</mp:DialogActivityDef>
```

> **Note:**
>
> In this case, you must supply a bean as input that includes the input parameters required by the dialog.
>
> ```
> <mp-link id="showHistory" params="label:
> getString('SHOW_HISTORY'),
>                     destination:
> cb(controller.showHistoryDialog,
>                                 bean('title', 'Title',
> 'metricName',
>                                     'CPUProcessorPerf',
> 'metricColumns',
>                                     ['CPUIdle'],
> 'timePeriod', 'LAST_DAY'))">
> </mp-link>
> ```
>
> For more examples, see the MetricHistoryDialog.html and the AvailabilityDialog.html files from the Demo Sample.

## Defining Trains and Train Pages

The train activity enables you to define a train (a guided workflow or wizard) by stringing together a series of pages.

To define a train, include a declaration of the train itself (`TrainActivityDef`) and each of the steps (`TrainStepActivityDef`) in the Integration class:

```
<mp:TrainActivityDef id='addNewUserEmbeddedTrain' label='Add New User'>
  <mp:stepActivities>
    <mp:TrainStepActivityDef id='anuStep1' label='User Info'
                  pageClass='dhs/user/UserInfo'
                  pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
    <mp:TrainStepActivityDef id='anuStep3' label='Credentials'
                  pageClass='dhs/user/Credentials'
                  pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
    <mp:TrainStepActivityDef id='anuStep2' label='Expiry'
                  pageClass='dhs/user/Expiry'
```

```
                                    pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
      <mp:TrainStepActivityDef id='anuStep4' label='Schedule'
                            pageClass='dhs/user/Schedule'
                            pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
      <mp:TrainStepActivityDef id='anuStep5' label='Notifications'
                        pageClass='dhs/user/Notifications'
                        pageControllerClass='dhs/user/
NotificationsTrainStepController'/>
      <mp:TrainStepActivityDef id='anuStep6' label='Confirmation'
                        pageClass='dhs/user/Confirm'
                        pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
  </mp:stepActivities>
</mp:TrainActivityDef>
```

The TrainController includes the following methods:

- `init(Train)`: a method that is called when the train is loaded, and enables you to control the model associated with the train.

- `trainDone`: a method that is called when the user clicks the **Finish** or **Cancel** button within the train. At that point, you can inspect the train state (whatever is stored in the train model) to do one of the following

  - Control if the train should complete and continue to the completion activity

  - Take some other action such as moving the train back to a previous step by using the `train.setStep` method or end the train and invoke another activity.

Each train step HTML within the train starts with the mp-train-step-page tag(a special type of Page) and be associated with a controller (`TrainStepController`). In this case, the controller is a special type of `PageController`, and includes support for the `init(Page)` method that enables you to initialize the contents of the train page. Because the page is within a train, it might refer to either its own page model (such as `model().property`) or it might refer to data stored in the train model (such as `train.model().property`).

Finally, in either the train step controller or the train controller, the code can check for state and if the train can complete, that is, all the required information is entered, then the controller code can call `train.setMayFinish()`.

## Defining URLs

UrlActivityDef support the ability to define a URL that can be accessed using the invokeActivity directive from a UI component click handler (for example, InfoItem, ImageLink, and Button). The URL can be represented as an absolute URL including all request parameters, or parameters can be supplied at runtime. To define the URL that should have URL parameters substituted at runtime, define the UrlActivityDef to include inputParams as follows:

```
 <mp:UrlActivityDef id='oracle' label='myExtApp' urlBase="http://www.extapp.com"
>
     <mp:inputParams>
                 <mp:InputParam
name='pageId' />

     </mp:inputParams>
 </mp:UrlActivityDef>
```

To reference the URL the invokeActivity directive used specifying the id of the
UrlActivityDef and passing a bean that includes the parameter and the appropriate
value. The parameters provided will be added to the URL as request parameters.

```
<mp-info-item id="currentLoad" params="label: 'CPU Load',
                     value: respData.result.getString('','Load'),
                     destination: function(){invokeActivity('extapp',
bean('pageId','Load'));}"
></mp-info-item>
```

In this example, the URL that is accessed is http://www.extapp.com&pageId=Load.

# Packaging the MPCUI Implementation With the Plug-in

Include the MPCUI implementation in a plug-in by placing a metadata definition of the
MPCUI in the /mpcui subdirectory of the plug-in stage directory. For information about
the structure and packaging of plug-ins, see Validating, Packaging, and Deploying the
Plug-in .

Put the MPCUI metadata file and any other files (CSS, JS, etc.) in the following
directories:

```
plugin_stage/oms/metadata/mpcui/my_mpcui_metadata.xml

plugin_stage/oms/metadata/mpcui/js/libs/mylib/my_mpcui.js
…
```

> **Note:**
>
> In the previous examples, set the names of the XML
> (my_mpcui_metadata.xml) and other files according to your requirements as
> a plug-in developer.

# Defining System Home Pages

For target types identified as system targets, there are three options for which home
page is rendered for the system target.

1. Display the Enterprise Manager default system home page.

   This page shows a summary of the availability and incidents for the system
   members. This option is enabled by either of the following:

   • Omitting MPCUI metadata from your plug-in

   • Including MPCUI metadata in the plug-in and including the following
     `<EmuiConfig>` element in the MPCUI metadata file:

   **Example: Using the Default System Home Page**

   ```
   <CustomUI target_type="demo_hostsystem"xmlns="http://www.oracle.com/
   EnterpriseGridControl/MpCui">

     <EmuiConfig>
         <use-framework-homepage>true</use-framework-homepage>
   ```

```
            </EmuiConfig>
</CustomUI>
```

**Figure 9-1    Default System Home Page**
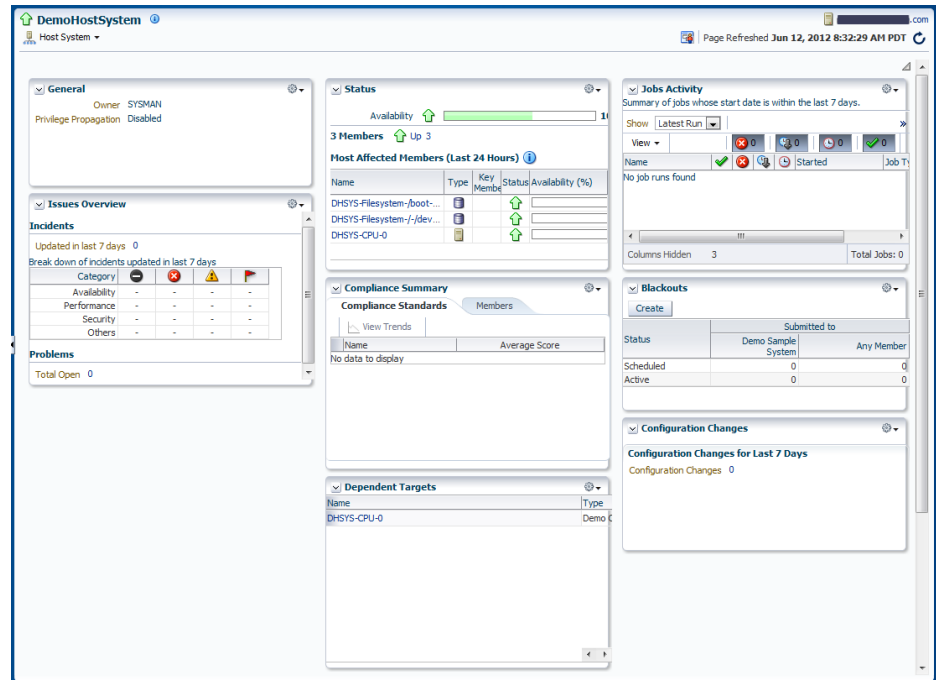


**2.** Display the Enterprise Manager default system home page, with some customized content.

The home page can show a number of prepackaged regions in a customized layout. The use of the default home page is controlled by metadata as illustrated in the example in step 1.

The selection of regions and their layout on the home page is specified by including systemUiIntegration metadata in the plug-in. For more information, see Defining systemUiIntegration Metadata

**Figure 9-2    System Home Page With Some Customization**
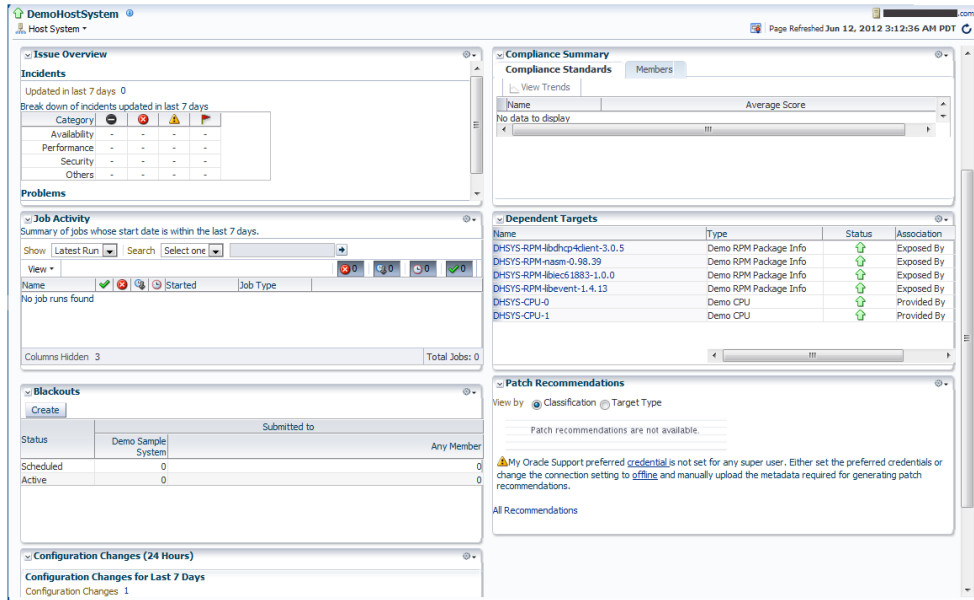


3. Construct a custom home page using the MPCUI capabilities included with the EDK.

   The home page is constructed using HTML/JS and MPCUI. There are several data services and UI components that are provided by MPCUI specific to system or composite target types. For more information, see Defining System Regions

**Figure 9-3    Customized System Home Page**
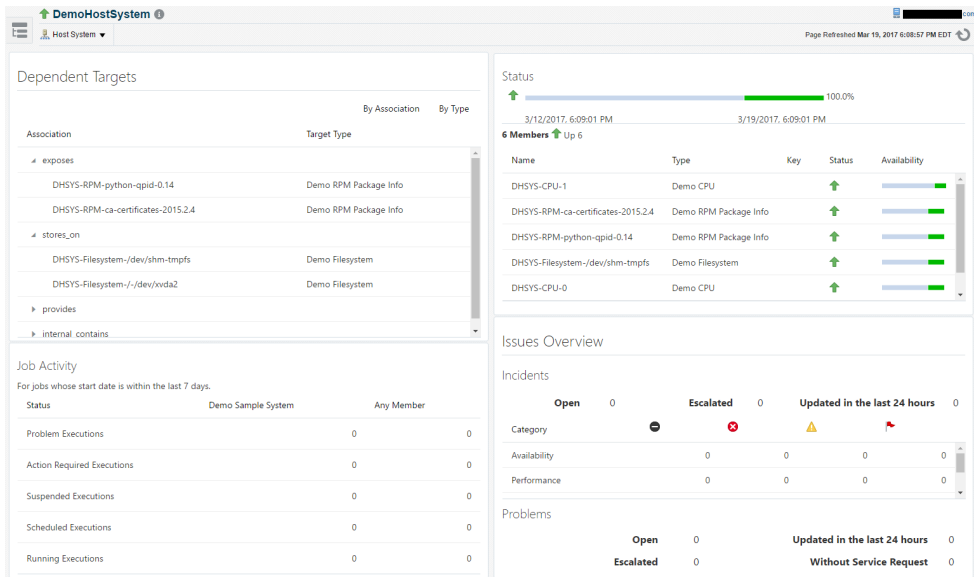


# Defining systemUiIntegration Metadata

To use the default system home page with some customized content:

1. Define a systemUiIntegration Metadata XML file for your target type including the following information:

- Preferred layout

- Add or remove regions (only required if you want to modify regions)

  The following example provides an example of a systemUiIntegration Metadata XML file.

  For information about the XML Schema Definition (XSD) that governs the systemUiIntegration Metadata XML file, see *ORACLE_HOME*/sysman/ emSDK/core/system/xml/SystemUiIntegration.xsd.

**Example: systemUiIntegration Metadata XML**

```
<systemUiIntegration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/
SystemUiIntegration.xsd"
        xmlns="http://www.oracle.com/EnterpriseGridControl/
SystemUiIntegration">

<general targetType="demo_hostsystem"
         defaultLayout="twoColumnNarrowLeft"
         showOptionalRegions="false"
         topLevelTarget="true"
         allowCreateFromSystemsUi="true"/>


 <region taskFlowId="/WEB-INF/db/system/region/db-system-region-hihgavail-
task-flow.xml#db-system-region-hihgavail-task-flow"
            titleResBundle="oracle.sysman.db.rsc.inst.DBMsg"
                                titleNlsId="GENERAL"
                                titleDefText="General"
                                regionType="add"
                                displayOrder="1" />

 <region taskFlowId="/WEB-INF/sdk/core/regions/events/console/incident-
overview-task-flow.xml#incident-overview-task-flow"
            titleResBundle="oracle.sysman.core.groups.ui.CoreGroupsUiMsg"
                                        titleNlsId="ISSUE_OVERVIEW"
                                        titleDefText="Issue Overview"
                                        regionType="add"
                                        displayOrder="4" />

 <region taskFlowId="/WEB-INF/sdk/core/regions/jobs/jobs-activity-task-
flow.xml#jobs-activity-task-flow"
            titleResBundle="oracle.sysman.db.rsc.inst.DBMsg"

titleNlsId="JOB_ACTIVITY"

titleDefText="Job Activity"
                                                        regionType="add"

displayOrder="7" />

 <region taskFlowId="/WEB-INF/db/system/region/db-system-region-dep-members-
task-flow.xml#db-system-region-dep-members-task-flow"
            titleResBundle="oracle.sysman.core.groups.ui.CoreGroupsUiMsg"

titleNlsId="DEPENDENT_TARGETS"

titleDefText="Dependent Targets"
```

```
                                                              regionType="add"

displayOrder="9" />

 <region taskFlowId="/WEB-INF/sdk/core/regions/gccompliance/target/
compliance-overview-task-flow-brief.xml#compliance-overview-task-flow-brief"
             titleResBundle="oracle.sysman.core.groups.ui.CoreGroupsUiMsg"

titleNlsId="COMPLIANCE_SUMMARY"

titleDefText="Compliance Standard Summary"
                                                              regionType="add"

displayOrder="6" />

 <region taskFlowId="/WEB-INF/sdk/core/regions/mos/patch/target-patch-
recommendation-task-flow.xml#target-patch-recommendation-task-flow"
             titleResBundle="oracle.sysman.db.rsc.inst.DBMsg"

titleNlsId="PATCH_RECOMMEND"

titleDefText="Patch Recommendations"
                                                              regionType="add"

displayOrder="12"/>

 <region taskFlowId="/WEB-INF/config/adfc/blackout/region/emcore-groups-
blackout-task-flow.xml#blackout_group_taskflow"
             titleResBundle="oracle.sysman.core.groups.ui.CoreGroupsUiMsg"

titleNlsId="BLACKOUTS"

titleDefText="Blackouts"
                                                              regionType="add"

displayOrder="2" />

 <region taskFlowId="/WEB-INF/sdk/core/regions/ecm/history/config-history-
task-flow.xml#config-history-task-flow"
             titleResBundle="oracle.sysman.db.rsc.inst.DBMsg"

titleNlsId="CONFIG_CHANGES"

titleDefText="Configuration Changes (24 Hours)"
                                                              regionType="add"

displayOrder="5" />

</systemUiIntegration>
```

2. Save the systemUiIntegration Metadata XML file to the following directory:

   *plugin_stage*/stage/oms/metadata/systemUiIntegration

3. If your plug-in is deployed already, then you can use the `emctl register oms metadata` command to update the MPCUI part of your plug-in only. For more information about the `emctl register oms metadata` command, see Updating Deployed Metadata Files Using the Metadata Registration Service (MRS).

# Defining System Regions

The MPCUI framework supports a number of regions that can be used as part of a home page built to display information for a system target.

## Defining System Status Region

The system status region shows the recent availability of the system target and all of its members. The region is included in the system home page by using the following tag:

```
<mp-status-overview-region id="statusOverview" height="50%"></mp-status-overview-region>
```

**Figure 9-4    System Status Region**
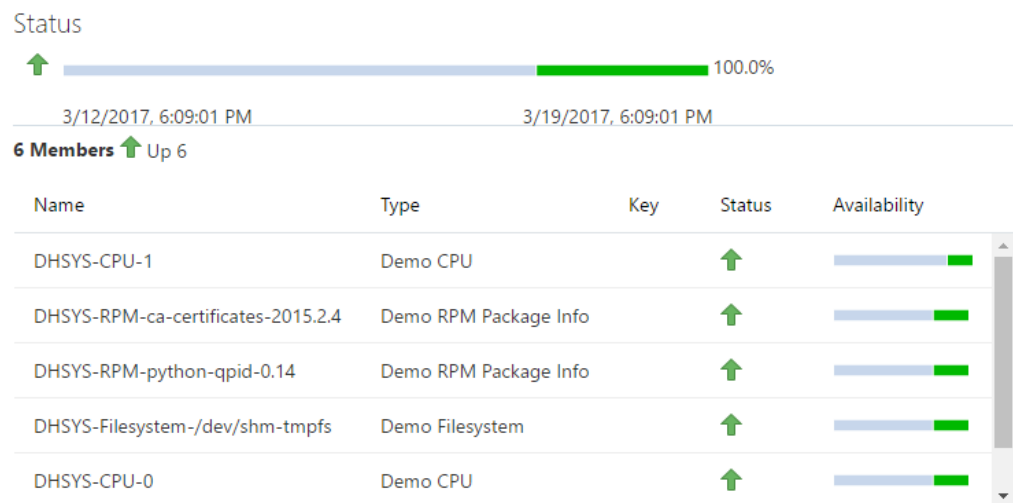


## Defining System Issues Region

The system issues region shows the summary count of incidents for all of the targets in the system. The region is included in the system home page by using the following tag:

```
<mp-issues-overview-region id="issuesOverview" height="50%"></mp-issues-overview-region>
```

Figure 9-5    Issues Overview Region



## Defining the System Job Activity Region

The system job activity region displays the number of jobs in each of the primary job status for the system target and the summary for all the system members.

```
<mp-jobs-activity-region id="jobsOverview" height="40%"></mp-jobs-activity-region>
```

Figure 9-6    System Job Activity Region



# Defining Navigation

Navigation in the MPCUI application can be either of the following:

- Between activities defined in the application. For more information, see Navigation to Activities.

- To other URLs, where URL refers to other Enterprise Manager pages or to external URLs. For example:

  ```
  http://www.example.com
  ```

  For more information, see URL and Links.

## Navigation to Activities

Defining Navigation describes the approach to navigating between activities. These descriptions apply to navigating to activities from the menu or from another activity defined in HTML..

This section describes how to navigate to another activity from within the controller code, that is the JavaScript code associated with an activity.

```
MyController.prototype.showProcessorHistory = function(event) {
    // show an example of invoking an activity (a dialog in this case) and
    // getting information from the dialog when it returns (is closed)

    // create the context to be passed to the dialog
    var bean = new Bean('targetName',
      TargetContext.getTargetName(), 'targetType',
      TargetContext.getTargetType(),
        'metric', 'CPUProcessorPerf', 'columns', ['CPUIdle'],
        'period', 'LAST_DAY', 'title', 'Metric History');

     this.page.invokeActivity('metricHistory', bean,

this.page.cb(this.processorHistoryDone));
  };
```

The preceding example shows a controller method that uses the `page.invokeActivity` method to redirect to another activity (in this case, a dialog).

> **✎ Note:**
>
> The callback input is this.page.cb(this.processorHistoryDone). The purpose of this notation is to maintain the context within the callback. When a function gets called, we guarantee the value of "this" by creating a closure. "this.page.cb" is a helper utility which binds the current "this" to your callback, so when the code finally gets into this.processorHistoryDone "this" has the same value, which would typically be your Controller class.

## URL and Links

There are a number of different methods for navigating from components in the MPCUI application to other locations through a URL. Use the `mp-link` component to render an HTML-style link including a tool tip and location.

- Absolute URL (external to Enterprise Manager)

  To provide a link to an absolute URL, use the "UrlAbs" class and an instance of this class can then be associated with a Link destination or can be accessed through the `invokeActivity` method.

**In the Page HTML:**

```
<mp-link id="gotoOracle" label="Oracle" destination="model().oracleUrl">

</mp-link>
```

**In the Controller Class:**
```
page.setModel("oracleUrl", new UrlAbs("http://www.oracle.com",
"Oracle"));
```

Alternative method using `invokeActivity`:

**In the Page Class:**
```
<button label="Go To Oracle"
        click="function(){invokeActivity(model().oracleUrl)}" />
<button label="Go To Oracle"
            click="invokeActivityCb(model().oracleUrl, null)" />
```

**In the Controller Class:**
```
page.setModel("oracleUrl", new UrlAbs("http://www.oracle.com",
"Oracle"));
```

- Link to Enterprise Manager Page Using Page Constants

  In addition to absolute URLs, the MPCUI framework supports the ability to link to well known Enterprise Manager pages by constructing a "UrlEm" object that can be referenced from the Link destination or passed to the `invokeActivity` method as part of a click handler. The reference guide includes a complete list in the oracle.sysman.emx.Constants class of all page constants available and the corresponding parameters that must be specified to produce a URL.

  ```
  // setup link to availability page
  var availLink:UrlEm = new UrlEm(Constants.PAGE_AVAILABILITY,
                                  [new InputParam(Constants.P_TARGET_NAME,
                                   ApplicationContext.getTargetName()),
                                   new InputParam(Constants.P_TARGET_TYPE,
                                   ApplicationContext.getTargetType()),
                                   new InputParam(Constants.P_PAGE_TYPE,
                                   Constants.BY_DAY)]);
  page.setModel("availPageLink", availLink);
  ```

- Link to Enterprise Manager Pages That Do Not Have Constants Defined

  Note that `UrlEm` can only be used to access pages that are supported via page constants in the oracle.sysman.emx.util.Constants class. For pages that do not currently have constants defined, you can access a page by creating a `UrlRel` object containing the page's ADF view ID value.

  For example, to access the Bare Metal Provisioning dashboard, you would specify the page's view ID (/faces/core-bmp-dashboard) as follows:

  ```
  var url:UrlRel = new UrlRel("/faces/core-bmp-dashboard", null);
  ```

  The easiest way to find the view ID for a given ADF page is in the page URL; it is the string following http://<server:host>/em/.

- Link to Enterprise Manager Target Home page

  A special case is to produce the URL to an Enterprise Manager target home page. For this situation, use the static `UrlEm.homepageUrl` method:

  ```
  page.setModel("relatedHostLink", UrlEm.homepageUrl(host.name, host.type));
  ```

- Dynamic URL Using "DIRECT_URL"

  For cases where a URL must be constructed dynamically at runtime from a
  data service, the following option may be used. The activity id "DIRECT_URL" is
  reserved for the special case and is provided by the framework. No UrlActivityDef
  is declared in this case, but instead the invokeActivity directive is passed a bean
  that specifies the "url" property. The value provided for that property will be used
  as the URL to direct to when the component is clicked.

  In the following example, the data service "respData" is queried to obtain a URL.
  This would be replaced by whatever data service is used within the page to obtain
  the necessary URL. This may be a MetricValuesDataService or a SqlDataService.

```
<mp-info-item id="currentLoad" params="label: 'CPU Load',
        value: respData.result.getString('','Load'),
        destination: invokeActivity('DIRECT_URL',
        bean('url',
                    respData.result.getString('','Load')))">
</mp-info-item>
```

## Adding Links to External Applications

Providing the ability to link to other applications outside of Enterprise Manager is not
currently supported.

# Accessing Enterprise Manager Data

The MPCUI framework provides access to Enterprise Manager services through
JavaScript interfaces to the Enterprise Manager Web services layer. You can access
these client services directly when necessary. Although in many cases, the services
are further abstracted through UI components that utilize them to interact with the
Enterprise Manager server to obtain the appropriate data to be displayed in the
management UI.

The following sections describe the various services included in the MPCUI framework
and provide brief examples of how these services can be used from your code.

> **Note:**
>
> The EDK does not support accessing arbitrary Web services. The
> appropriate way to access Web services would through the Management
> Agent residing on the service host, as either metrics, jobs or remote
> commands invoked by a fetchlet.

## Metric Services

The MPCUI provides a simple service for retrieving metric data from the Management
server in either real-time or historical form. For real-time data, the Oracle Management
Service accesses the Management Agent to retrieve the data, so use this for cases
where the metric can be collected efficiently in real time.

## Using the Metric Values Service Transparently

Usually the metric values service is used transparently from a chart by specifying the metric to be displayed in the chart and in the case of a line chart, the periodicity of the data.

```
<mp-chart id="cacheChart" style="width:100%;height:100%"
                          params="mpChart: {
                             type: 'line',
                             metricName: 'MSSQL_MemoryStatistics',
                             metricColumns: ['cache_hit_ratio'],
                             timePeriod: 'REALTIME',
                             interval: 15}" >
 </mp-chart>
```

In this case, the caller never interacts directly with the service. The MPCUI framework uses the service to retrieve the data for the chart.

In the case of the table component, you can specify the metric directly also:

```
<mp-table id="processesTable" style="width:100%;height:100%"
              params="mpTable: {metricName: 'CPUProcessesPerf',
              metricColumns: ['ProcUser', 'ProcCPU', 'ProcCmd'],
              timePeriod: 'REALTIME', interval: 30,
              columns: [
                 {headerText: 'Key', field: 'key', id: 'key', headerStyle:
'width:50px'},
                 {headerText: 'User', field: 'ProcUser', id: 'ProcUser',
headerStyle: 'width:100px'},
                 {headerText: 'CPU', field: 'ProcCPU', id: 'ProcCPU',
headerStyle: 'width:80px'},
                 {headerText: 'Command', field: 'ProcCmd', id: 'ProcCmd',
headerStyle: 'width:400px'}
              ]}"
              >
</mp-table>
```

## Using the MetricValuesDataService Tag

Use the `mp-metric-values-data-service` tag within a page (or dialog) to display metric data in a table component, where the dataService attribute of the table is set to the data service. Then the data from the metric service is displayed in the table or when data from the service will be shared between multiple components (for example, the table and a link or label).

**Example: Using the MetricValueDataService Tag**

```
    <mp-data-services>
      <mp-metric-values-data-service id="mv1" params="flattenData: true
                 targetName: appModel.target.name,
                 targetType: appModel.target.type,
                 metricName: 'Load', columns: ['cpuUtil', 'cpuUser',
'cpuKernel'],
                 timePeriod: 'LAST_DAY'"
                 > </mp-metric-values-data-service>
    </mp-data-services>
    <mp-table id="mvTable" params="mpTable: { dataservice: 'mv1' }" > </mp-table>
```

# Calling the Metric Value Service from a Controller

The metric value service can be called from within a controller. This is the most flexible means of using the service and allows the caller to manipulate the data as necessary before adding the final results to the model so that it can be displayed in the UI.

**Retrieving Individual Values from the Metric Service (HTML)**

You can retrieve individual values from the metric service in order to display them in a Label, InfoItem, or other such component.

```
<mp-metric-values-data-service id="procData" params="flattenData:true,
    targetName:appModel.target.name,
    targetType:appModel.target.type,
    metricName:'CPUProcessorPerf',
    columns:['CPUIdle'],
    timePeriod:'CURRENT',
    interval:15">
</mp-metric-values-data-service>
```

Then from the component that will display the value:

```
<mp-info-item params="label: 'CPU(0) Idle %',
    value="procData.result.getString('0','CPUIdle')">
</mp-info-item>
```

**Example: The Metric Service from a Controller**

```
var cpuPerf =
        TargetContext.getTargetContext().getMetric("CPUPerf");
var cpuPerfSel = procMetric.getSelector(
        ['system','idle', 'io_wait']);
cpuPerfSel.getData(page.cb(this.cpuDataHandler),
            MetricCollectionTimePeriod.CURRENT, page.getBatchRequest());
```

Use the metric service by creating a `MetricSelector` for a particular metric, and then calling the `getData` method on that selector. When calling the `getData` method, two parameters are passed:

* a callback to the handler that will be called with the result of the request

* the periodicity of the selection

When the service request has completed, either successfully or with an error, the handler is called and passed the results of the request and a fault. The caller must check for the presence of the fault before proceeding with any processing of the data result.

**Example: Metric Service Result Handler**

```
MyController.prototype.cpuDataHandler = function(cpuData, fault) {
 if (fault != null) return; // handle this better!

 var dataPoint = cpuData.results[0];
 var collectionTime = dataPoint.timestamp;
 var idleTime = dataPoint.data[0]['idle'];
 var systemTime = dataPoint.data[0]['system'];
 var ioWaitTime = dataPoint.data[0]['io_wait'];
};
```

To access the data, you must have the reference to
`dataService.result.getString('key','column')`. The key is required to identify the
row in the sample to be returned in cases where the metric supports multiple keys. If
the metric does not include a key column, then the key value should be passed as '' or
null. The column is the data column to be retrieved from the metric definitions.

Each data point (`TimestampMetricData`) has a time stamp member that tells you when
that data point was collected, and includes a data array that is effectively a table for
that metric.

If the metric has multiple keys (such as process, file systems, and so on), then the
data array has multiple rows, one for each key, and each row has the requested data
columns. In the previous examples, the data array contains one row for each process.
If your metric does not include key columns, then the data array contains a single row
only.

Each row in the data array is a `KeyMetricData` object. If your metric has keys, then the
`metricKey` property tells you to which key the row applies. If you have no key for your
metric, then ignore this property. The `KeyMetricData` is a dynamic object into which
you can index, using the column name to get the value for that column.

In the previous examples, the code walks the rows in the data array, and for each
row (`KeyMetricData`) it gets the `ProcUser` column from the data. The original request
also included the `ProcCPU` and `ProcCmd` columns, so those could be accessed in
the same way, that is, `data['ProcCPU']` or `data['ProcCmd']`.

## Metric Data Source Filters

When using the metric data service through the mp-metric-values-data-service in
HTML or the MetricSelector in JavaScript, it may be useful to request that the set of
data returned by the service be filtered according to some additional selection criteria.
This can be accomplished within the controller by implementing a custom data source
and then filtering the results of the metric service in the controller and populating the
custom data source with the results.

It is also possible to define a metric filter that can be applied to the request which will
cause the service itself to filter the results and return only the filtered set to the client
for display.

The metric filter, referred to as MetricPredicate, is made up of several elements,
including individual column filters, the filter operator, and the optional order by criteria.
Each column filter specifies a column to filter, the operator to filter by. and a value to
filter against. The column filters support typical operators for numeric data, including
GT, LT, GE, and LE.

For string data, the operators include EQ, NE, and REGEX. The REGEX operator
will perform a regular expression string match using each value with the filter input
value as the pattern. The regular expression pattern match is done using Java regex
libraries, so the pattern should conform to the requirements of Java pattern matching.

The predicate operator combines the column filters into a single expression and
supports either an AND (all column filters must be satisified) or an OR (any of the
column filters being satisfied is sufficient). The order by criteria specifies a column to
order the results by and a row count to limit to. This is useful in cases where a "top-N"
result is desired.

When constructing a metric filter, the columns filters can be optional and an order by
only specified. Alternately, the order by can be optional and the column filters only are

specified. When constructing a metric filter, all columns included in column filters and in the order by must be part of the same metric, and it must be the metric that is being selected in the corresponding metric data service request. Combining columns from multiple metrics into the same filter is not supported.

The following example describes the process for defining a metric filter on a mp-metric-values-data-service tag. The data service tag includes the "predicate" property which is bound to the corresponding metric filter (MetricPredicate) as such:

```
<mp-metric-values-data-service
  id="fsMetDs"
  params="metricName: 'FilesystemPerf',
    columns: ['MountPoint','Utilization','FreeKB','UsedKB','TotalKB','FSType',
  'FSName'],
    targetName: appModel.target.name,
    targetType: appModel.target.type,
    timePeriod: 'LAST_HOUR',
    predicate: model().fsFilter"
  ></mp-metric-values-data-service>
```

In the controller associated with the page, the filter is constructed by specifying the filter columns, operator, and order by criteria. In the following example, the file systems metric request from the service above is filtered to those filesystems with a TotalKB size of greater than 1000kb and a regular expression match on the filesystem name (FSName) of '.net'. Finally, the results are ordered by the FreeKB column descending limited to the first five filesystems.

```
MyController.prototype.createFsFilter = function() {
  var filters = [
                new MetricFilter('TotalKB', MetricOperator.GT, 1000),
                new MetricFilter('FSName', MetricOperator.REGEX, '(.*)net(.*)')
                ];
  var orderBy = new MetricOrderBy('FreeKB', MetricOrderBy.DESC, 5);
  var predicate = new MetricPredicate(filters, MetricOperator.AND, orderBy);

  return predicate;
};
```

## Custom Data Source

In addition to the metric and SQL data sources (and service tags) that can be used to obtain data for charts, tables and other components, you can construct your own custom data source for these components. This is useful in situations where you want to obtain data from other MPCUI services and manipulate it before display. For example, to combine data from two metrics, filter the data in some way, or otherwise aggregate the data.

Creating a custom data source requires the use of controller code to obtain the source data and then to manipulate it to create the data source. The custom data source provides the following important behavior:

- Set column descriptors for the data included in the data source to provide help to the UI component when displaying the data. The descriptor contains properties such as data type, and display label (for legends or column headers).

- Support multiple data points to enable the display of the data in a time-series chart.

- Support caching and modification of the data source allowing components to show updated data as information underlying the data source changes.

## Creating the Custom Data Source

Typically the custom data source (`oracle.sysman.emx.model.CustomDataSource`) is constructed and set in the page model using `Page.setModel`. When constructing the data source, you must specify the columns (or data items) that make up the data source along with a flag that can indicate the following:

- If the data should be treated as if it includes a key

  Specify the key only if the data source will be displayed in a chart that honors keys such as a bar or column chart. If the data will be shown in a tabular view or a non-chart component, then you do not have to identify one of the columns as a key.

- If the data should be treated as if it includes multiple timestamp samples

  Specify that the data includes timestamps only if the data will be displayed in a time-series chart (`LineChart`) and might have data samples added to the data source over time by using the MPCUI polling mechanism.

```
function CustomDataSource(columns, hasKey, isTimeSeries)
```

The Array of columns specifies the data items included in the data source. This array can be either:

- an array of strings, with each string specifying the label of the data item

- an array of column descriptors (either `QueryColumDesc` or `CustomColumnDesc`

  Specifying a column descriptor enables you to specify a label for the column and a data type (for `QueryColumnDesc`) or to specify additional properties to display the data in a tabular display such as the column width, that is, if the column is sortable, and so on (for `CustomColumnDesc`).

The following example shows a result handler in the controller that is set up to handle data returned from a request to the `SqlQueryService`.

**Example: Handling Data Returned From a Request to the SQLQueryService**

```
// execute a SQL query and then massage the data for display
            var query = new SqlQueryService('CPU_USAGE',
                        [SqlQueryInput.createParam("TARGET_GUID",
                        TargetContext.getTargetContext().guid)]);
            query.execute(page.cb(this.cpuQueryHandler),
page.getBatchRequest());
        }


    ProcessesPageController.prototype.cpuQueryHandler = function(result, fault) {
        var page = this.page;
        if (fault != null || result.getError() != null) {
            MpLog.logError(fault, "Getting CPU Data via SQL Query");
            return;
        }

        var cpuSqlData = page.getModel("cpuSqlData");
        if (cpuSqlData == null) {
            cpuSqlData = new CustomDataSource([
```

```
                    new QueryColumnDesc("Processor", QueryColumnType.STRING),
                    new QueryColumnDesc("Idle Percentage",
QueryColumnType.DECIMAL),
                    new QueryColumnDesc("Used Percentage",
QueryColumnType.DECIMAL)
                    ], true);
            page.setModel("cpuSqlData", cpuSqlData);
        }

        var rows = result.rows;
        if (rows != null) {
            for (var r=0; r<rows.length; r++) {
                var id = result.getString(r, 'CPU Number');
                var idle = result.getNumber(r, 'Idle %');
                var used = result.getNumber(r, 'Used %');
                cpuSqlData.setRow("Processor #"+id, idle, used);
            }
        }

    };
```

In the previous example, the data source is constructed with three columns and the data types are specified. The second parameter to the constructor is passed as true, indicating that the data should be treated as if it has a key. In this case, the first column in the list is always treated as the key. You cannot specify a different position in the data.

Finally, for each row in the SqlQueryResultSet (`result.rows`), the code constructs a row in the custom data source.

> **✎ Note:**
>
> For a complete working example, see the
> `demo_hostsample,ProcessesPageController.js` in the EDK.

## Binding the Data Source to a UI Component

In the page layout (for example, ProcessesPage.html), the data is bound to the UI component using the `customDataSource` property. In the following example, note `cpuSqlTable`. This is a table that displays the data loaded into the `cpuSqlData` custom data source.

**Example: Binding the Data Source to a UI Component**

```
<mp-table id="cpuSqlTable" params="mpTable: { customDataSource:
model().cpuSqlData, rootAttributes: { style: 'width:100%;height:100%' } }">
</mp-table>
```

Figure 9-7 shows what the previous example displays.

**Figure 9-7    Table Displaying Data Loaded into the cpuSqlData Custom Data Source**



## Updating the Custom Data Source

Because the data source is bound to the UI component, when you update it, the UI displays the new data automatically. You have two options to update a custom data source:

1. Call either the `CustomDataSource.setRow` or `setRows` methods.

   These methods are used when you have a data source that does not include timestamped data. In this case, you are modifying the row or rows included in the data source.

2. If the data source includes timestamped data, then call the `CustomDataSource.setTimestampedRows` method.

   This method adds a new sample to the time series and typically is used in the case where the data source is displayed in a line chart. Adding a new sample by calling this method causes a new time slice to appear on the line chart.

For more information about these methods, see the API Reference and the demo_hostsample for examples using the Custom Data Source.

## Computed Data Source

The Enterprise Manager metric collection framework supports the ability to compute values from counters. However, the values are only guaranteed to be correct when retrieved from the historical data collected by the agent and stored in the repository. Attempting to query these values from a real-time request (for example, MetricValuesDataService with timePeriod set to REALTIME) can lead to unexpected results as the value computed utilizes the last counter stored during historical collection and not a counter stored for the real-time collection. As such, if you require realtime display of computed metrics you may need to consider using the computed data source.

The computed data source provides the ability to combine data from two metrics into a single display. This is useful when the metric to be displayed is based on a compute expression using a stored counter a described previously.

To use this data source, you typically define two metrics. One metric computes the values to be collected and stored in the repository for historical purposes. This metric includes the compute expressions that consume the stored counters. The other metric would be a transient metric that only collects the counters themselves. This metric would not be collected for historical purposes as the raw counter values are typically not useful.

You need both sets of metrics when constructing the computed data source in the UI code. The first metric, which specifies the metrics in their computed form, is called the "source" metric. The data source uses these metrics to define the display attributes for the data, including the labels for the columns, and will also retrieve any required historical data.

```
/**
* Construct a data source that shows the CPU-System% and CPU-Idle% from
historical
* data and then appends data to it from a real-time data source that acquires
* counter columns and derives the values from the counters.  First declare the
* columns to be shown on the chart, the labels will be based on the metric-
column * labels and will obtain the historical data that initially populates the
chart.
*/
var srcCols = [
    new ComputeColumnDesc( TargetContext.getTargetContext(), "CPUPerf",
    "system"),
    new ComputeColumnDesc( TargetContext.getTargetContext(), "CPUPerf",
    "idle"),
    ];

/**
* These are the counter columns; they do not need to be from the same metric as
* the source columns, however the counter columns must be from the same metric
as
* all other counters.
*/
var ctrCols = [
    new ComputeColumnDesc( TargetContext.getTargetContext(), "CPUPerf",
    "systemCounter"),
    new ComputeColumnDesc( TargetContext.getTargetContext(), "CPUPerf",
    "idleCounter"),
    ];

/**
* create the data source and pass the source columns, the counter columns and a
* pointer to the compute function.  Finally pass the page the data source will
be
* consumed on and the interval to be used to populate the data.  The compute
* function will be called at each interval.
*/
var computedDataSource = new ComputeDataSource(srcCols, ctrCols,
                         page.cb(this.computeFunction),
                         page, PollingInterval.EVERY_15_SECONDS);
page.setModel("compDataSource", computedDataSource);
```

The computed data source will then send a request for the historical data, and will then begin polling for the counters at the interval specified. Each time a sample is retrieved, the compute function will be called and passed a reference to the computed data source and a data point (TimestampMetricData) that contains the latest set of values for the counter metrics.

The compute function can then compute values using the counters and must return a data point that contains the metrics with the same name as those that were identified in the source columns. In the previous example, the counter columns are "systemCounter" and "idleCounter", but the data point that is returned from the compute function must include a value for the source columns, "system" and "idle".

```
MyController.prototype.computeFunction = function(ds, dp) {
    // retrieve the counter values from the data point passed; could also retreive
    // any necessary context from the data source
    var systemCounter = dp.data[0]["systemCounter"];
    var idleCounter = dp.data[0]["idleCounter"];

    // compute values; this is where you would replicate the logic in your
    // computed metric
    var systemValue = systemCounter+Math.floor(Math.random()*(50 - 20 + 1)) + 20;
    var idleValue = idleCounter+Math.floor(Math.random()*(120 - 80 + 1)) + 80;

    // you must now return a TimestampMetricData object.  You can use the one
passed and return
    // it, but to do so you must add columns to the data point.  The index
reference [0] is
    // a reference to the fact that the datapoint could have multiple rows, one
for each key
    // but the example does *NOT* support multiple keys.  Also, if you created a
new
    // data point to return you would need to set the timestamp of the datapoint
    // correctly, using the timestamp of the sourced datapoint
    dp.data[0]["system"] = systemValue;
    dp.data[0]["idle"] = idleValue;

    return dp;
};
```

# Packaged SQL and the Query Service

While the MPCUI framework provides access to the most useful data through either UI components or simplified services (such as the metric service), inevitably you must have access to other information stored in the Management Repository in a more unstructured form. The MPCUI framework provides a SQL query service for this access.

The SQL query service enables you to package SQL statements with your plug-in and then run the statements through a Web service and then bind that data to UI elements in your custom UI. The SQL query service does not provide an open-ended or scriptable API to the Management Repository as this would expose a potential security risk.

The SQL query service can only run SQL statements that have been deployed to the Management repository through the Enterprise Manager Extensibility Framework. This ensures that the statements can access EDK views only. This still provides you with a lot of flexibility and the ability to access data from your own views (for example, views generated from Enterprise Manager configuration data) along with Enterprise Manager partner EDK views.

You can encapsulate the query service entirely within the page code by using the `mp-sql-data-service` tag. This tag allows the caller to specify the SQL to be processed and the parameters to be passed. This data service object can then be bound to a table or to other UI components that support it.

**Example: Using the SQLDataService Tag**

```
<mp-data-services>
    <mp-sql-data-service id="dbSummaryDS" params="queryID: 'DATABASE_SUMMARY',
                properties: model().dbSummProp">
    </mp-sql-data-service>
  </mp-data-services>
  <mp-table id="dbSummaryTable" params="mpTable: { dataservice:
'dbSummaryDS' }" > </mp-table>
```

**Retrieving Individual Values From the SQL DataService**

To reference a specific cell returned from `SQLdataService` for use within a component (such as Link or Label), the following type of reference is used:

```
<mp-sql-data-service id="ids" params="queryID: 'INSTANCE_INFO',
                properties: props('TARGET_GUID',appModel.target.guid)" ></mp-
sql-data-service>

    ...

 <mp-info-item params="label: 'CPU Model',
                    value: ids.result.getString(0,'CPU Model')"></mp-info-
item>
```

The reference to the data service is through `dataService.result.getString(rowIndex,'column')`, where `rowIndex` is the row returned from the query and `column` is the name of the column as specified in the original SQL query.

The query service can also be called from within a controller, providing much more flexibility in terms of how the data is manipulated before it is displayed. There are two APIs that provide access to the query service:

- SqlQuery interface

  The SqlQuery interface allows for a single SQL query to be processed, passing the bind variable and receiving a result set in return. The result set provides an interface quite similar to that of the JDBC ResultSet.

  **Example: Using the SqlQuery API**

  ```
    var getInfoSvc = new SqlQuery("GET_TARGET_INFO",
            [["TARGET", name],["TYPE", type]]);   // bind variables
        getInfoSvc.execute(page.cb(this.getTargetInfoHandler));

  MyController.prototype.getTargetInfoHandler = function(resultSet, fault)
  {
        var target;
        if (fault != null) {
            if (resultSet != null && resultSet.getError() == null) {
                target.setGuid(resultSet.getBase64Binary(0, "TARGET_GUID"));
                target.setTypeMetaVer(resultSet.getString(0, "TYPE_META_VER"));

                var props = [];
                for(var i=1; i<Target.NUM_PROPERTIES+1; i++)
                    props.push(resultSet.getString(0, "CATEGORY_PROP_"+i));
                target.setCatProperties(props);
            }
      };
  ```

- BulkSqlQuery interface

  The bind variables are referenced by name and correspond to the variables as represented in the packaged SQL statement:

  ```
  SELECT target_guid, type_meta_ver, category_prop_1, category_prop_2,
         category_prop_3, category_prop_4, category_prop_5
   FROM mgmt_targets
   WHERE target_name = ?TARGET?
   AND target_type = ?TYPE?
  ```

  When a number of queries can be processed in a single request, you can use the `BulkSqlQuery` interface. Each query must be added to the bulk query and when all queries to be processed have been added, the `BulkSqlQuery.execute` method is called and passed the result handler that will be called with the results.

  When a result handler for the `SqlQuery` is passed a single `SqlQueryResultSet` for the processed query, the result handler for the `BulkSqlQuery` is passed a `BulkResultSet`. Then it must retrieve the `SqlQueryResultSet` for each query using the request id specified when the query was added.

  A separate request id is required to support the case where the same query can be processed multiple times with different bind variables as part of the same bulk request.

  **Example: Using the BulkSqlQuery API**

  ```
  var guidParam = [["TARGET_GUID", TargetContext.getTargetContext().guid]];

    var bulkQuery = new BulkSqlQuery();
    bulkQuery.addQuery("INSTANCE_INFO", "INSTANCE_INFO", guidParam);
    bulkQuery.addQuery("PROCESS_STATES", "PROCESS_STATES", guidParam);
    bulkQuery.addQuery("PROCESS_INFO", "PROCESS_INFO", guidParam);

    bulkQuery.execute(page.cb(this.pageDataHandler), page.getBatchRequest());

  MyController.prototype.pageDataHandler = function(bulkResult, fault) {
    var info = bulkResult.getResultSet("INSTANCE_INFO");
  ```

## Guidelines for Writing Packaged SQL

Adhere to the following guidelines when writing packaged SQL for the MPCUI:

- Packaged SQL can only access views that are part of the partner EDK.
  This includes any views that are generated as a result of configuration metric definitions.

- Any SQL that attempts to modify data (update or delete) will be filtered by the MRS during plug-in deployment.

- SQL statements that attempt data definition language (DDL) will be filtered out by the MRS and are not allowed

- Anonymous PL/SQL (for example, begin, end constructs) are not allowed as access to PL/SQL procedures is not allowed from packaged SQL

- Bind variables must be identified by a text identifier and prefixed and suffixed by a `?`. For example, `?TARGET_TYPE?`

- Bind variables are not case sensitive

- The query service restricts the size of result sets to 1000 rows or 100,000 bytes, so care should be taken to limit the size of the possible result set returned by a query.

## Packaging SQL in the Plug-In

SQL Statements used in the MPCUI code are packaged with the MPCUI metadata using the `SqlStatements` element

For information about the location of SQL statements in the MPCUI metadata, see Overview of MPCUI Metadata Elements. For information about the MPCUI metadata XSD, see the EDK Metadata API reference.

## Getting Target Type Information

For cases where the plug-in UI requires information about a related target type, such as its display name, but does not require the details about a specific instance of that target type, the TargetFactory provides a function to retrieve this summary information.

The TargetFactory.getTargetTypeInfo function returns a TargetTypeInfo object that contains the display name of the target type. When calling this function, pass a TargetTypeInfo object with the internal targetType provided (for example, "oracle_database) and a handler function. The handler will be called with the TargetTypeInfo and any fault that occurred during the processing of the request:

```
var typeInfo = new TargetTypeInfo("oracle_database");
            TargetFactory.getTargetTypeInfo(typeInfo,
page.cb(this.getTypeInfoHandler));
    }

    MyController.prototype.getTypeInfoHandler = function(typeInfo, fault) {
      if (fault != null) {
        MpLog.logError(fault, "Getting Target Type Info");
        return;
      }

      MpLog.info("Target Display Label for (oracle_database):"+
                typeInfo.typeDisplayName);
    };
```

## Working With Target Services

In addition to the services described previously, the MPCUI framework provides a number of other services that are an integral part of the Target object (`oracle.sysman.emx.model.Target`). When the MPCUI application is running, the `TargetContext.getTargetContext()` call returns the Target instance for the primary target.

You can construct other target instances for associated targets. In either case, use the following methods to obtain additional information for these targets through the MPCUI service layer.

## Target Properties Service

For any instance of the Target class, you can call the `getTargetInfo()` method to retrieve the target properties associated with that target instance. The returned

target information populates the properties of the Target instance including: `guid`, `catProperties`, `typeMetaVer`, `timezoneRegion`, and so on.

For information about these properties, see the Target class documentation in the EDK (/doc/partnersdk/mpcui/emcore/doc/oracle/sysman/emx/model/Target.html).

When calling the `getTargetInfo()` method, you must provide a handler. This handler will be called when the `targetInfo` service returns. It is passed the fully populated Target instance and a fault object that is set to include any errors that occurred during the processing of the request to retrieve target properties:

```
var target = TargetContext.getTargetContext();
    target.getTargetInfo(page.cb(this.targetInfoHandler));


        MyController.prototype.targetInfoHandler = function(target, fault)
```

> **Note:**
>
> In the case of `TargetContext.getTargetContext()`, the current target information is loaded when the application starts and it is not necessary to call `getTargetInfo()` for that target instance unless you think that target properties have changed.

## Associated Targets Service

Use the `target.getAssociatedTargets()` method to retrieve the set of targets related to a target instance. This method is called and passed an array of association types and a handler that is called with the list of associated targets. Refer to the API documentation for a full description of the types of the objects returned by this method:

```
// get associated host
    var target = TargetContext.getTargetContext();
    var assocTypes = [ AssociationDataService.HOSTED_BY ];
    target.getAssociatedTargets(assocTypes, page.cb(this.assocHandler));

        MyController.prototype.assocHandler = function(assocResult, fault) {
            var host = assocResult.getAssoc(AssociationDataService.HOSTED_BY);
            if(host != null)
                page.setModel("relatedHost", host.name);
        };
```

## Metric Metadata Service

Use the `target.getMetricMetadata ()` method to retrieve the metric definitions information for a target instance. The metric metadata information is retrieved by calling the `Target.getMetric()` method which returns a Metric object for a specified metric name. Refer to the API documentation for a full description of the types of the objects returned by this method:

```
var target = TargetContext.getTargetContext();
    target.getMetricMetadata(Util.createProxy(this, this.metadataHandler));


 MyController.prototype.metadataHandler = function(target, fault)
```

> **Note:**
>
> In the case of `TargetContext.getTargetContext()`, the current target metric metadata is loaded when the application starts and it is not necessary to call `getMetricMetadata()` for that target instance unless you think that target metadata has changed (which is unlikely).

## Availability Service

Use the `target.getAvailability()` method to retrieve current availability information for a target instance. The availability information (`AvailOverviewData`) includes the current status, the up time (%) for the last 24 hours and so on. Refer to the API documentation for a full description of the types of the objects returned by this method:

```
var target = TargetContext.getTargetContext();
        target.getAvailability(page.cb(this.targetAvailHandler));


        MyController.prototype.getTargetAvailHandler = function(availInfo, fault)
```

## Automated Polling of Service Requests

> **Note:**
>
> An important use of the "REALTIME" data selection for any chart, table, or data service is that it initiates automated polling of the data at the specified interval.

The MPCUI framework supports a limited subset of intervals (15, 30, 45, 60 seconds) so that requests can be grouped together to avoid a large number of requests to the Management Server.

The MPCUI framework starts and stops the polling of these requests automatically as each page or dialog appears or is removed (goes out of scope).

You cannot initiate a polling request that is persistent beyond the scope of a page or dialog.

## Batching of Service Requests

In addition to the batching of polling requests, the MPCUI framework provides the ability to explicitly batch requests made at runtime from activity (page or dialog) controllers. Batching of requests is a good practice as it avoids additional round trips to the Management Server which slows the performance of your UI pages and adds additional overhead to the Management Server.

The most common opportunity to batch requests is as part of the activity initialization.

- For data services declared in the page layout (HTML file), the MPCUI framework will batch the requests for you.

- For service requests you make from your controller.init() method, you can pass the page's batch request to the service methods. The MPCUI framework calls the init() method after your page is loaded.

The following example is extracted from the HomePageController.js file in the Demo Sample. Note the instances of page.getBatchRequest() in the method. All requests made in this way will be performed over a single pass to the Management Server.

**Example: Batching Requests as Part of the Activity Initialization**

```
MyController.prototype.init = function(pg) {
    this.page =
pg;

    var target = TargetContext.getTargetContext();

    var guidParam = [["TARGET_GUID", target.guid]];
    var bulkQuery = new BulkSqlQuery();
    bulkQuery.addQuery("INSTANCE_INFO", "INSTANCE_INFO", guidParam);
    bulkQuery.addQuery("CPU_USAGE", "CPU_USAGE", guidParam);
    bulkQuery.execute(this.page.cb(this.queryResultHandler),
                        this.page.getBatchRequest());

    // get processes metric to get process summary information
    var procMetric = target.getMetric("CPUProcessesPerf");
    var procSelector = procMetric.getSelector(['ProcUser', 'ProcCPU',
'ProcCmd']);
    procSelector.getData(this.page.cb(this.processesHandler),
            MetricCollectionTimePeriod.CURRENT, this.page.getBatchRequest());

    var cpuPerf = target.getMetric("CPUPerf");
    var cpuPerfSel = cpuPerf.getSelector(['system', 'idle', 'io_wait']);
    cpuPerfSel.getData(this.page.cb(this.cpuDataHandler),
            MetricCollectionTimePeriod.REALTIME,
this.page.getBatchRequest());

    // get associated host
    var assocTypes = [ AssociationDataService.HOSTED_BY ];
    target.getAssociatedTargets(assocTypes, this.page.cb(this.assocHandler),
            this.page.getBatchRequest());
};
```

You can use batch requests elsewhere in controller code by creating a MultiServiceRequestor (batch request) and passing it to each request made. For example, suppose that in response to a button click in the page, two requests will be made to the Management Server to retrieve information. They could each be made separately (resulting in two trips to the server) as shown in the following example:

**Example: Creating Individual Batch Requests**

```
var procMetric = TargetContext.getTargetContext().getMetric("CPUProcessesPerf");
var procSelector = procMetric.getSelector(['ProcUser', 'ProcCPU', 'ProcCmd']);
 procSelector.getData(this.page.cb(this.processesHandler),
            MetricCollectionTimePeriod.CURRENT);    // 1st round trip

 var cpuPerf = TargetContext.getTargetContext().getMetric("CPUPerf");
 var cpuPerfSel = cpuPerf.getSelector(['system', 'idle', 'io_wait']);
 cpuPerfSel.getData(this.page.cb(this.cpuDataHandler),
            MetricCollectionTimePeriod.REALTIME);  // 2nd round trip
```

Alternatively, you can combine the batch requests into a single batch request avoiding the additional round trip to the Management server as shown in the following example:

**Example: Combining Batch Requests**

```
var batchRequest = new MultiServiceRequestor();

var procMetric = TargetContext.getTargetContext().getMetric("CPUProcessesPerf");
var procSelector = procMetric.getSelector(['ProcUser', 'ProcCPU', 'ProcCmd']);
  procSelector.getData(this.page.cb(this.processesHandler),
           MetricCollectionTimePeriod.CURRENT, batchRequest);

var cpuPerf = TargetContext.getTargetContext().getMetric("CPUPerf");
var cpuPerfSel = cpuPerf.getSelector(['system', 'idle', 'io_wait']);
  cpuPerfSel.getData(this.page.cb(this.cpuDataHandler),
           MetricCollectionTimePeriod.REALTIME, batchRequest);

 batchRequest.sendRequest();      // 1st and ONLY round trip!
```

> **Note:**
>
> You must call the `sendRequest()` method to commit the batch request. Otherwise, no requests will be sent. In the case of the `PageController.init` use of `page.getBatchRequest()`, this is *not* necessary because the MPCUI framework will do it for you.

# Software Library Search Service

When the plug-in UI requires information about Software Library entities, it can search using different criteria, including name, status, maturity level, or entity attributes. The desired entities can be filtered by specifying the query criteria using the SearchField enumeration. A list of EntityInfo objects that represent an entity revision that match the query criteria is returned. The URN in the EntityInfo object can be used as a unique identifier for the entity revision. If any error has occurred, it will be set in ListSwlibEntitiesResult.errorMessage.

```
// search the software library
    var swSearch = new ListSwlibEntities();
    swSearch.addSearchInput(new SearchInput(SearchField.NAME, "oracle"));
    var swlib = Swlib.getSwLib();
    swlib.listEntities(swSearch, Util.createProxy(this, this.swSearchHandler));
  }

  MyController.prototype.swSearchHandler = function(result, fault) {
    var r, e;
    if (fault != null) {
      MpLog.logError(fault, "Search Software Library");
       return;
    }

     for (var i=0; i<result.swlibEntitiesList.length; i++) {
       var entity = result.swlibEntitiesList[i];
       MpLog.info("Swlib Entity: "+entity.toString());
      }
```

```
   page.setModel("swlibContents", result.swlibEntitiesList);
};
```

# Performing Task Automation

The following sections describes how to perform task automation with examples.

It includes the following:

- Automation Services
- Working With Credentials

## Automation Services

One of the more powerful aspects of the MPCUI framework is the ability to provide access to administrative features through a UI customized to that purpose. The framework supports the processing of administrative tasks through the Enterprise Manager job system and Web services that provide access to the job system.

The MPCUI provides the following job services:

- `Job.submit`

- `Job.runSynchronous`

- `JobExecution.getStatus`

- `JobExecution.getDetails`

- `JobExecution.stopJob`

- `JobExecution.deleteJob`

- `JobExecution.getJobDetailsURL`

- `RemoteOp.performOperation`

## Submitting or Running a Job

The job service allows any job that is registered with the plug-in target type to be submitted for processing. The service does not support the ability to submit system job types at this time.

Scheduling of jobs through the job service supports a limited set of the scheduling options supported by the job system. The job schedule supports the following options:

- Immediate, once, hourly, daily, weekly, monthly, yearly
- Start and end time for repeat submissions
- Repeat count and frequency
- Starting period grace time
- Management Repository or target time zone

Supported job parameter types include Vector, Scalar, Large and ValueOf.

As with other services, requests are issued asynchronously. This requires that a handler is provided that will be called when the request has completed (or failed). When submitting a job, the result handler is called and passed a JobExecution object.

This object contains the processing context for the job that was submitted, and can be used to retrieve the status of the job and operate on the job (stop or delete it).

### Example: Submitting a Job

```
var job = new Job("backup", "MyBackup", null,
                  TargetContext.getTargetContext(),
                  [Job.jobParam("dsn", "AdminDS"), Job.jobParam("sql_cmd",stmt)],
                   JobSchedule.IMMEDIATE);
  job.submit(this.page.cb(this.jobSubmitHandler));
}

MyController.prototype.jobSubmitHandler = function(exec, fault) {
        // using exec (JobExecution) can now get current status of job,
        // get step details, and start or stop the job
  var execId = exec.getExecutionId();
};
```

When a job is run in this way (using the submit method), the job is submitted for processing and the service returns immediately. Therefore, the status of the job may change from submitted to running, and then to complete and the client must check the status periodically.

The job service also provides a way to submit a job for immediate processing and will wait (synchronously) until the job execution completes, fails or reaches a timeout. The client handler will not be called until this state is reached.

### Example: Running a Synchronous Job

```
var job = new Job("backup", "MyBackup", null,
            TargetContext.getTargetContext(),
            [Job.jobParam("dsn", "AdminDS"), Job.jobParam("sql_cmd",stmt)],
            JobSchedule.IMMEDIATE);
    job.runSynchronous((this.page.cb(this.jobRunHandler), 30); // 2nd param is
timeout
}

MyController.prototype.jobRunHandler = function(exec, fault) {
    // using exec (SynchronousJobExecution) can get details about job execution;
    // this handler will not be called until the job completes, fails
    // or the timeout is reached
    var execId = exec.getExecutionId();
};
```

The Task interface is a simplified way of submitting a job for immediate processing, without requiring all of the additional settings associated with the Job.submitJob API.

### Example: Using the Task API

```
var task = new Task("TTisql", null, [Job.jobParam("dsn", "AdminDS"),
                       Job.jobParam("sql_cmd",stmt)]);
            task.execute(this.page.cb(this.createTableHandler), 10);  // timeout
is 10s
        }

 MyController.prototype.createTableHandler = function(result, fault) {
            var status = result.getRunStatus();
            if (status == JobStatus.RUNNING) {
                // timed out while waiting for job... still running
```

In the case of a synchronous job, the status of the job is available immediately from the result passed to the handler; however, it should be checked to see if it equals `JobStatus.RUNNING`. If so, then the request reached the specified timeout and the caller must treat the job execution as if it were submitted asynchronously.

# Getting Job Status and Step Details

After a job has been submitted, there are several APIs available to get the status of the job and the details of each step including job output. To use these APIs, the caller must have a valid `JobExecution` object, which is passed to the result handlers of `submit` and `runSynchronous` APIs. Currently, there is no service provided that allows a client to search for a job execution.

**Example: Getting Job Status**

```
MyController.prototype.submitHandler(exec, fault) {
    exec.getStatus(Util.createProxy(this, this.statusHandler));
};

MyController.prototype.statusHandler(status, fault) {
    if(status.getStatus() === JobStatus.FINISHED)
```

Getting the job status for a submitted job requires service request, and therefore requires a handler to be called with the result (and possibly a fault if the request processing fails). In addition to the status of the job, the job step details can be retrieved.

**Getting Job Details**:

Use the `JobExecution` object that is passed to the submit handler, to retrieve step output details as well as the job status. If the job has failed or if the step has not completed, then no data will be returned.

In the case of a synchronous job execution, the handler for the `Job.runSynchronous` or `Task.execute call` can check the job status and if complete, retrieve the job details from the result directly:

```
MyController.prototype.createTableHandler = function(result, fault) {
    var status = result.getRunStatus();
    if(result != null && result.Status() === JobStatus.COMPLETED) {
        var steps = result.getStepDetail();
        for(var i=0; i<steps.length; i++) {
            var detail = steps[i];
            proc.addDetailText(detail.output);
        }
```

In the case of an asynchronous job execution, the handler for the `Job.submit` handler must call `JobExecution.getStatus`, and then `JobExecution.getDetails`. Each call requires a request to the server:

```
MyController.prototype.submitJob = function() {
    var params = [];
    params.push(Job.jobParam("p0","p0value"));
    var job = new Job(type, name, desc, TargetContext.getTargetContext(),
                    params, Job.immediateSchedule());
    job.submit(this.page.cb(this.submitHandler));
};

MyController.prototype.submitHandler = function(result, fault) {
```

```
        if(fault == null && result != null) {
            // get the job status; calls the server
            result.getStatus(oj.Object.createCallback(this, this.statusHandler));
        }
   };

   MyController.prototype.statusHandler = function(result, fault) {
       if(fault == null && result != null) {
           if(result.getStatus() === JobStatus.COMPLETED) {
               // now can get job output details

result.getJobExecution().getDetails(this.page.cb(this.detailsHandler));
           }
       }
   };

   MyController.prototype.detailsHandler = function(result, fault) {
       if(fault == null && result != null) {
           var steps = result.getStepDetail();
           for(var i=0; i<steps.length; i++) {
               var detail = steps[i];
               proc.addDetailText(detail.output);
           }
       }
   };
```

## Using a Timer to Periodically Check Job Status

If a job is submitted asynchronously (`Job.submit`) or runs synchronously but the request reaches a timeout and returns a job status of `JobStatus.RUNNING`. This indicates that the job is still running, and you might want to check the status of the job again at a later point.

The easiest thing from a code perspective is to expose a UI element that the user interacts with to cause the application to check the status of the job. For example, the UI might show a *Running* indicator with a button or link labeled **Check Status Now**. When the user clicks the button or link, it calls the `JobExecution.getStatus` method to retrieve the updated status.

If the required interaction pattern is that the UI remains active while the job is running in the background, and periodically updating the UI with information about the status of the job, then the MPCUI provides a job API to perform period checking of the job status. Each update calls a handler to provide the caller with the opportunity to process the current status and update the UI with that information.

## Stopping and Deleting a Job

Jobs submitted through the MPCUI APIs can be stopped or deleted using the following APIs:

### Example: Stopping a Job

```
  MyController.prototype.stopJob = function(exec) {
      // NOTE: the JobExecution must be a valid job context obtained from
submitted a job
      exec.stopJob(this.page.cb(this.stopJobHandler));
  };

  MyController.prototype.stopJobHandler = function(exec, fault) {
```

```
    if(fault == null && exec != null) {
        // job was successfully stopped
    }
};
```

**Example: Deleting a Job**

```
MyController.prototype.deleteJob = function(exec) {
    // NOTE: the JobExecution must be a valid job context obtained from
submitted a job
    exec.deleteJob(this.page.cb(this.deleteJobHandler));
};

MyController.prototype.deleteJobHandler = function(exec, fault) {
    if(fault == null && exec != null) {
        // job was successfully deleted
    }
};
```

For jobs that are submitted using the `Job.runSynchronous` API, the job can be deleted when completed by passing the `deleteWhenFinished` parameter as true. It is the third parameter and it defaults to false:

```
var job = new Job("backup", "MyBackup", null,
                  TargetContext.getTargetContext(),
                  [Job.jobParam("dsn", "AdminDS"), Job.jobParam("sql_cmd",stmt)],
                  JobSchedule.IMMEDIATE);
                  job.runSynchronous((this.page.cb(this.jobRunHandler), 30, true);
```

# Remote Operations

Using a job to perform administrative tasks is the most flexible approach in terms of scheduling and control (start, interrupt, or stop), but does come with the additional overhead of managing the task being processed. For simple tasks that do not require control over schedule and that are expected to be performed quickly, use the `RemoteOp` service.

This service allows the execution of a script packaged with the plug-in to be executed directly through the Management Agent.

> **Note:**
>
> The script must be packaged with the plug-in in the agent/ scripts directory (as described in the following section), and might require credentials or parameters to be processed.

**Packaging Scripts for Remote Operation**

Scripts included in a plug-in for remote operations must be included in the staging area:

```
stage/agent/scripts
```

You can create additional subdirectories under /scripts if required. Scripts placed in this location can be referenced using the `RemoteOp` service by referencing the `%scriptsDir%` variable. For example:

**Plug-in Stage Area**

```
./stage/agent/scripts/process/kill_process.pl
```

**MPCUI Code (JavaScript)**

```
var params = [
            RemoteOp.param("%scriptsDir%/process/kill_process.pl"),
            RemoteOp.param(pid) ];
    var remoteOp:RemoteOp = new RemoteOp("perl", params);
    remoteOp.performOperation(this.page.cb(this.killProcessHandler));
```

In this example, a RemoteOp object is constructed using the shell / command to run and the parameters to pass into that shell. The first parameter must always be the location of the script to be run, referencing its location relative to the %scriptsDir% variable. Subsequent parameters are included as required for the script being run.

To run the remote operation, the RemoteOp.performOperation method is called and passed a function that will be called when the remote operation completes processing. This handler has the following signature:

```
  MyController.prototype.killProcessHandler = function(remoteOp, fault)
```

If the remote operation failed to be communicated to the Management Agent, then the fault parameter will include the details of that error. If the remote operation was processed, then the fault will be null and the remoteOp parameter supplied.

Check the remoteOp parameter status because it can indicate an error status returned during command execution on the Management Agent. The following example shows this check being performed.

**Example: Checking the remoteOp Parameter Status**

```
/**
  * Check status; could be any number of problems some of which may result
  * in step output, some of which (like missing creds) result in a non-
successful
  * run status but no step details.
  *
  * result.getRunStatus() - the status of the job, refer to
Constants.JOB_*_STATUS
  * result.getStepDetail().stepName/detail - name and output from each step in
the job
  * result.getJob() - the complete job Object, to reference parameters:
  *   result.getJob().parameter[0].paramName/paramValue[0]
  *
  */
  if(remoteOp.result.status !== Constants.JOB_COMPLETED_STATUS) {
    // job did *NOT* complete successfully
    var pid = remoteOp.getParameter(2).paramValue[0];
    var msg = "Unable to successfully kill process ["+pid+
              "].  The status of the command was: "
             +Util.getCatString(Util.JOB_STATUS, remoteOp.result.status)
             +"\nReturn Code: "+remoteOp.result.returnCode
             +"\nCommand Output: "+remoteOp.result.commandOutput;
    MessageAlert.show(msg, "Failed to Kill Process", Alert.OK);
  } else {
    // successful job execution; process was killed; can look at the
    // step details to get possible output from the job
    MpLog.debug("Command was successful: "
```

```
         +"\nReturn Code: "+remoteOp.result.returnCode
            +"\nCommand Output: "+remoteOp.result.commandOutput);
      var tableId = $
("#processesTable").data(CustomElement.MPCUI_CHILD_NODE).getChildNodeId();
            // Get the table object
      var ojt = $("#" + tableId).data(MetricDataBinding.BINDING_DATA);
      ojt.refreshImmediate();
   }
```

# Working With Credentials

The Enterprise Manager credentials model supports three different modes for
performing operations that require credentials:

- Preferred Credentials

  Specific credentials are set for a target or all targets of a particular type. In this
  mode, the user does not select a set of credentials or provide credential values.

- Named Credential Set

  Sets of credentials are created for a target or all targets of a particular type, and
  each set is assigned a name. In this mode, the user is presented with a list of
  named sets and can select the set that they would use to perform the operation.

- Override Credentials

  In this mode, the user can supply credentials at runtime that are used to perform
  the operation.

# Retrieving Credential Information

MPCUI provides the facilities for retrieving credential information about a particular
target. The services can return information only that the current user is privileged to
see. This ensures that there is no unauthorized access to secure information. It also
requires that you must handle a situation where credential information might not be
available to the user accessing the MPCUI code.

## Check for Preferred Credentials

To check if a target has preferred credentials set for it, call the
`CheckPreferredCredsService.getPreferredCredsInfo` method as follows:

```
var ccSvc = new CheckPreferredCredsService();
var target = TargetContext.getTargetContext();
      ccSvc.getPreferredCredsInfo(target.name, target.type,
      'HostCreds', this.page.cb(this.checkPrefCredsHandler));
```

The service returns a `CheckPreferredCredsResult` object, which indicates whether
global (applying to all target instances of the type) or instance (applying only to the
single target instance) credentials are available:

```
  MyController.prototype.checkPrefCredsHandler = function(result, fault) {
    if(fault != null)
        MessageAlert.show(fault.faultDetail, "Error Checking Preferred Creds");
    else {
        var msg = "Checked for Preferred Credentials for
                target("+TargetContext.getTargetName()+","+
                TargetContext.getTargetType()+" for set(HostCreds)
                user("+ApplicationContext.getEmUser()+") \n"+
```

```
                         "Global Set = "+result.globalSet+" Instance Set = "
                          +result.instanceSet;
               MessageAlert.show(msg, "Check Preferred Creds Result");
        }
};
```

> **Note:**
>
> If preferred credentials are set, you can submit a job or perform a remote
> operation without passing any credentials information. In this case, the
> preferred set will be used.

## Retrieve Named Credentials Sets

You can retrieve the named credentials sets available for a particular target and:

- display the named credentials set in a choice (list or combo box)
- select from the named credentials set based on a convention required by your
  plug-in

The following code requests all named sets for two different credentials types for the
current target, and calls the `credSetResultHandler` handler with the result:

```
 var target = TargetContext.getTargetContext();
  target.getCredentialSets(["HostCreds", "HostSampleCreds"],
                  this.page.cb(this.credSetResultHandler));
```

The results handler can then consume the named sets return as appropriate (in this
example, constructing a data source for display in a table):

```
var credTableData;
   if(creds.credSet != null) {
      credTableData = creds.credSet.slice(0);

      // check to see if there are sets for both types
      var hostFound = false;
      var sampFound = false;
      for(var c=0; c<creds.credSet.length; c++) {
         if(creds.credSet[c].credentialType === "HostCreds")
             hostFound = true;
         else if(creds.credSet[c].credentialType === "HostSampleCreds")
             sampFound = true;
      }

      var missingType = ( !hostFound ? "HostCreds" : "HostSampleCreds" );
      var empty = new CredentialSet();
      empty.credentialType = missingType;
      empty.name = "<No Sets Found>";
      empty.guid = "";
      credTableData.push(empty);
   } else {
      empty = new CredentialSet();
      empty.credentialType = "<No Credential Sets Defined>";
      empty.name = "";
      empty.guid = "";
      credTableData = [empty];
   }
```

## Passing Credentials to Jobs and Remote Operations

This section discusses passing preferred credentials and named set credentials to jobs and remote operations.

**Preferred Credentials**

If the task (job or operation) to be performed attempts to use preferred credentials, then the credentials parameter passed to the task is omitted. Both the job and remote operation services will attempt to perform the task using preferred credentials. If no preferred credentials are set, then an error will be returned

**Named Set**

To specify that a named set be used to perform a task, the credentials are passed in a `JobCredential` (for jobs) and an `OpCredential` (for remote operations). In both cases, the credentials object includes the following four properties that must be set:

- `targetName`: the target the credentials apply to, usually `TargetContext.getTargetName()`

- `targetType`: the type of the target the credentials apply to, usually `TargetContext.getTargetType()`

- `usage`: the credentials usage as defined for the operation (see the job type definition). This usage specifies which credentials types are required and where they are applied during job execution

- `credGuid`: the identifier of the named set to be used. This is one of the properties of the `CredentialSet` class, which holds named credential sets. For more information, Retrieving Credential Information.

## Reusable Credentials UI Components

MPCUI provides a credentials region that may be included in a page to allow the end user to interact with the Enterprise Manager credentials subsystem to view the set of credentials available and to select preferred, named, or override credentials when performing a task (job or remote operation).

**Figure 9-8    Credentials Region**
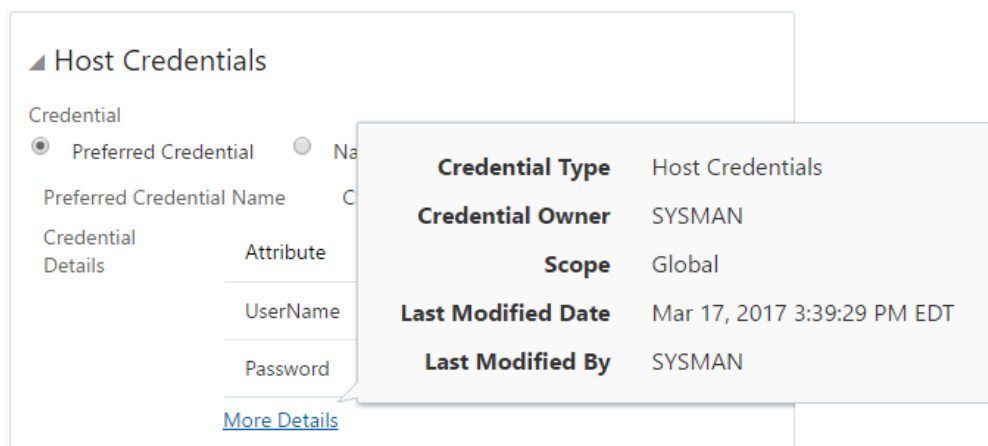
To include this region in a page, add the following HTML:

**Example: Adding a Credentials Region**

```
<mp-credentials-region id="creds" style="width:40%;height:100%"
        params="targetName: appModel.target.name, targetType:
appModel.target.type,
        credentialType: 'HostCreds'" ></mp-credentials-region>
```

From the page controller associated with the page, retrieve the settings applied by the end user to this region:

**Example: Retrieving Selected Credential Information**

```
MyController.prototype.getCredsEntered = function(event) {
    var creds = Util.def($
("#creds").data(CredentialsRegionBinding.BINDING_DATA),
                         null);
    if (creds == null) {
            // This notation is for use with the <mp-credentials-display> tag
      var credsId = $
("#creds").data(CustomElement.MPCUI_CHILD_NODE).getChildNodeId();
        creds = $("#" + credsId).data(CredentialsRegionBinding.BINDING_DATA);
    }
    var mode = creds.getMode();
    var msg = "Credential Option Selected: "+mode+"\n";

    var namedSet;
    var credentials;
    if (mode === CredentialsRegion.NAMED_MODE) {
      namedSet = creds.getNamedSet();
      msg += "Named Set Selected: "+namedSet;
    } else if (mode === CredentialsRegion.OVERRIDE_MODE) {
      try {
          credentials = creds.getOverrideCredentials();
          for(var c=0; c<credentials.length; c++)
            msg += "Field:"+credentials[c].label+", "+credentials[c].value+"\n";
      } catch(e) {
        msg += "Error Entering Credentials:\n";
        msg += e.message;
      }
    } else {
              // preferred selected...
    }
    MessageAlert.show(msg, "Credentials Entered");
  };
```

In the previous example, note that the mode determines if the user selected preferred, named, or override credentials. Depending on the mode, the named set can be retrieved (`CredentialsRegion.getNamedSet()`) or the override credentials can be retrieved (`CredentialsRegion.getOverrideCredentials()`).

## Managing Monitoring Credentials

The Target class provides the ability to retrieve and set the monitoring credentials for the current target. To retrieve the monitoring credentials, an instance of the Target class is required and the getMonitoringCredentials function is called, passing the results handler that will receive the credential meta data including the monitoring credentials set:

```
// get monitoring credentials
target.getMonitoringCredentials(this.page.cb(this.getMonCredResultHandler),
                                this.page.getBatchRequest());
```

The handler would appear as follows:

```
 MyController.prototype.getMonCredResultHandler = function(cred, fault) {
    if(fault != null) {
      if(cred != null && cred.isMissingCredentials()) {
               // no monitoring credentials are set
         MpLog.info("Monitoring Credentials have not been set:
"+fault.faultDetail);
      } else
           MpLog.logError(fault, "Get Monitoring Credentials");
      return;
    }

       /**
        * The CredentialTypeMetadata returned includes the meta-data for the
        * credentials as well as the actual values.  NOTE: credentials never
        * return the actual values for any field identified as a password it only
        * returns the masked "****" value.  You should never have any need to
        * access the actual values for a password field as any time credentials
        * are passed you are passing a credential set and don't need the actual
        * values of a pre-existing credentials set
        */

     var credFieldValues = cred.credentialSets[0].columnValues;
     for(var i=0; i<credFieldValues.length; i++) {
       var credField = credFieldValues[i];
       MpLog.debug("Monitoring Credentials["+credField.label+"] =
"+credField.value);
     }
   };
```

To set the monitoring credentials, the credentials fields according to the credential type specified for monitoring credentials. This is defined in your target metadata.

```
/**
* the CredentialSet passed contains the monitoring credentials to be set.
* Note that only the columnValues property of the credentials needs
* to be set when updating monitoring credentials as the framework
* derives the values for the credential set and type.  It is CRITICAL
* that the label set for each columnValue is the column NAME and not
* the display label for that column.  The name is the identifier assigned
* to the credential column in the target meta-data.
*
* In the demo_hostsample, for example, the credentials fields are:
*     name: SampleCredUser        label: User Id
*     name: SampleCredPassword    label: Password
*     name: SampleCredRole        label: Role
*/

var monitoringCreds = new CredentialSet();
monitoringCreds.columnValues = [
    new CredentialColumnValue("SampleCredUser", "myMonitoringUser"),
    new CredentialColumnValue("SampleCredPassword", "myMonitoringPassword"),
    new CredentialColumnValue("SampleCredRole", "myMonitoringRole")
    ];

var target = TargetContext.getTargetContext();
```

```
target.setMonitoringCredentials(monitoringCreds, Util.createProxy(this,

this.setMonCredResultHandler));
```

Then the handler would appear as:

```
MyController.prototype.setMonCredResultHandler = function(cred, fault) {
  if (fault != null) {
    MpLog.logError(fault, "Set Monitoring Credentials");
    return;
  }

      /**
      * if the set monitoring credentials was successful then the handler is
      * called with no fault and the set of credentials that were passed in
      */
};
```

# Storing Session State

The session state service provides the ability to store global state associated with the custom UI. This is useful for cases where state should be maintained for the current user, even as they move between pages outside of the HTML pages that define the custom UI. For example, if the user modifies the state of the home page and then navigates to the "All Metrics" page, and then upon returning to the home page you wish to restore the state of the page as the user left it. Because the user has left the pages that make up the MPCUI custom UI, it is necessary to store the state required on the server-side session established for this user session and not within the HTML/JS itself.

To set the session state, call the EmUser.setSessionData function, passing a SessionAttributes object. The session attributes contain an array of SessionAttribute objects, each of which has a corresponding name-value pair for the attributes stored.

```
CollectItemPageController.prototype.setSessionAttributes = function(modifier) {
        var page = this.page;

        var sessionData = new SessionAttributes();
        var item1Value = page.getModel("item1Value");
        var item2Value = page.getModel("item2Value");
        sessionData.attributes.push(new SessionAttribute("attr1", item1Value));
        sessionData.attributes.push(new SessionAttribute("attr2", item2Value));
        EmUser.setSessionData(sessionData,
page.cb(this.setSessAttrResultHandler));
    };

    CollectItemPageController.prototype.setSessAttrResultHandler =
function(attr, fault) {
        if (fault != null) {
            MpLog.logError(fault, "Set Session Data");
            return;
        }
    };
```

To retrieve the session state, use the corresponding EmUser.getSessionData service function. This function will retrieve the session state requested by passing a list of SessionAttributes and a handler that will be called with the result. This will be the same SessionAttributes, populated with the data retrieved from the session:

```
CollectItemPageController.prototype.getSessionAttributes = function(modifier) {
  var sessionData = new SessionAttributes();
  sessionData.attributes.push(new SessionAttribute("attr1"));
  sessionData.attributes.push(new SessionAttribute("attr2"));
  EmUser.getSessionData(sessionData,
this.page.cb(this.getSessAttrResultHandler), modifier);
};

CollectItemPageController.prototype.getSessAttrResultHandler = function(attr,
fault) {
  var page = this.page;

  if (fault != null) {
    MpLog.logError(fault, "Get Session Data");
    return;
  }
  for (var i=0; i<attr.attributes.length; i++) {
    var item = attr.attributes[i];
    if (item.name === "attr1") {
      page.setModel("lastItem1Value", item.value);
    } else if (item.name === "attr2") {
      page.setModel("lastItem2Value", item.value);
    }
  }
};
```

# Defining Page Layout Components

To ensure that the MPCUI page resizes correctly when the browser window resizes, Oracle recommends the following guidelines for page layout of an MPCUI-based page:

- Use the mp-row and mp-column containers

- The height and width can be set in percentage sizes.

- mp-row and mp-column use the **flexible boxes (or flexbox) layout standard**, so you may also use flexbox settings to layout your components

For example, to create a layout of three rows, each occupying one third of the height of the page, then enter the following in the HTML file:

**Example: Defining a Page Layout of Three Rows**

```
<mp-column style="height:100%;width:100%">
        <!-- 1st row -->
        <mp-row style="height:33%;width:100%">
        </mp-row>

        <!-- 2nd row -->
        <mp-row style="height:33%;width:100%">
        </mp-row>

        <!-- 3rd row -->
        <mp-row style="height:33%;width:100%">
        </mp-row>
 </mp-column>
```

Then enter the following to split each row horizontally into two separate or equal sections:

**Example: Splitting Rows into Two Equal Sections**

```
<mp-column style="height:100%;width:100%">
        <!-- 1st row -->
        <mp-row style="height:33%;width:100%">
            <mp-column style="height:100%;width:50%" >
            </mp-column>
            <mp-column style="height:100%;width:50%" >
            </mp-column>
        </mp-row>

        <!-- 2nd row -->
        <mp-row style="height:33%;width:100%">
            <mp-column style="height:100%;width:50%" >
            </mp-column>
            <mp-column style="height:100%;width:50%" >
            </mp-column>
        </mp-row>

        <!-- 3rd row -->
        <mp-row style="height:33%;width:100%">
            <mp-column style="height:100%;width:50%" >
            </mp-column>
            <mp-column style="height:100%;width:50%" >
            </mp-column>
        </mp-row>
 </mp-column>
```

Within each section, include individual components to fill out the layout of the page.

> **Note:**
>
> When using percentages to layout your page, it is important to remember to set the height and width as a percentage at every level in the hierarchy. Otherwise, the HTML will get confused about what container to which the percentage applies.

## Defining Panels

A panel is a visual box with a title that can be expanded and collapsed. For example, in the Defining a Page Layout of Three Rows example, each of the rows could be split up into separate panels rather than more vertical containers:

**Example: Defining Panels**

```
<mp-column style="height:100%;width:100%">
     <!-- 1st row -->
     <mp-row style="height:33%;width:100%">
         <mp-panel style="height:100%;width:50%" params="title: 'Row 1 Region
1'" >
         </mp-panel>
         < mp-panel style="height:100%;width:50%" params="title: 'Row 1 Region
2'" >
         </mp-panel>
     </mp-row>

     <!-- 2nd row -->
     <mp-row style="height:33%;width:100%">
         <mp-panel style="height:100%;width:50%" params="title: 'Row 2 Region
```

```
1'" >
        </mp-panel>
        < mp-panel style="height:100%;width:50%" params="title: 'Row 2 Region
2'" >
        </mp-panel>
    </mp-row>

    <!-- 3rd row -->
    <mp-row style="height:33%;width:100%">
        <mp-panel style="height:100%;width:50%" params="title: 'Row 3 Region
1'" >
        </mp-panel>
        < mp-panel style="height:100%;width:50%" params="title: 'Row 3 Region
2'" >
        </mp-panel>
    </mp-row>
 </mp-column>
```

The previous example results in a display similar to Figure 9-9. You can use each
of the panels to contain other UI components (such as tables and charts) to display
meaningful information. For more detailed examples, see the examples in the Demo
Sample included in the Extensibility Development Kit.

**Figure 9-9    Panels**



| Row 1 Region 1 | Row 1 Region 2 |
| Row 2 Region 1 | Row 2 Region 2 |
| Row 3 Region 1 | Row 3 Region 2 |

# Including Packaged Regions

The MPCUI supplies several packaged regions that can be included in your page with
a single simple tag.

# Availability Region

The availability region displays the availability of the target for the period specified in
the mp-availabilty-region tag daySpan property. It shows a segmented bar that shows
details of the target availability (such as outages) over that same period:

```
<mp-availability-region style="width:25%;height:100%" params="daySpan:1">
</mp-availability-region>
```

**Figure 9-10    Availability Region**
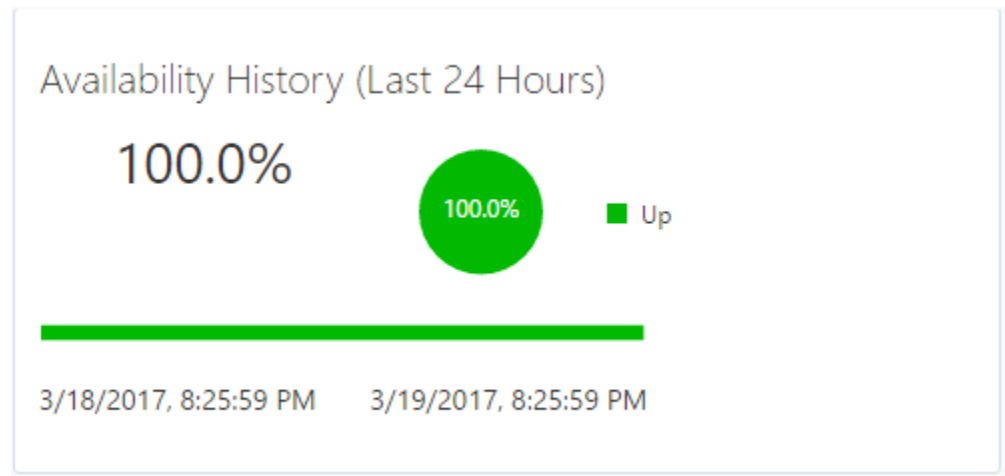


## Incidents and Problems Region

The incidents region shows the set of open incidents for the current target and all related targets. It provides the option to filter the list of displayed incidents. The only necessary settings for the region are the size (width/height):

```
<mp-incident-region style="width:50%;height:100%"></mp-incident-region>
```

**Figure 9-11    Incidents and Problems**
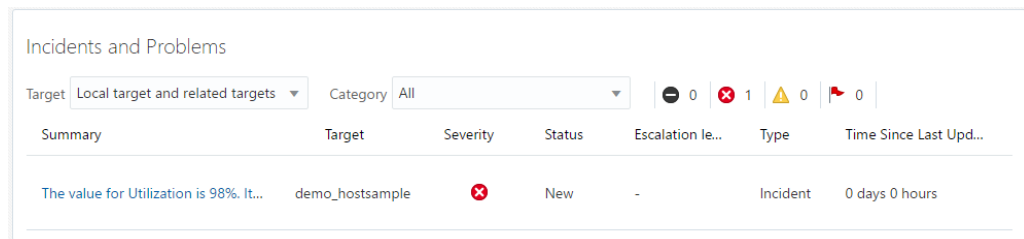


## Job Summary Region

The jobs summary region displays the count of jobs by status.

```
<mp-job-summary-region style="width:20%;height:100%"></mp-job-summary-region>
```

**Figure 9-12    Job Summary**
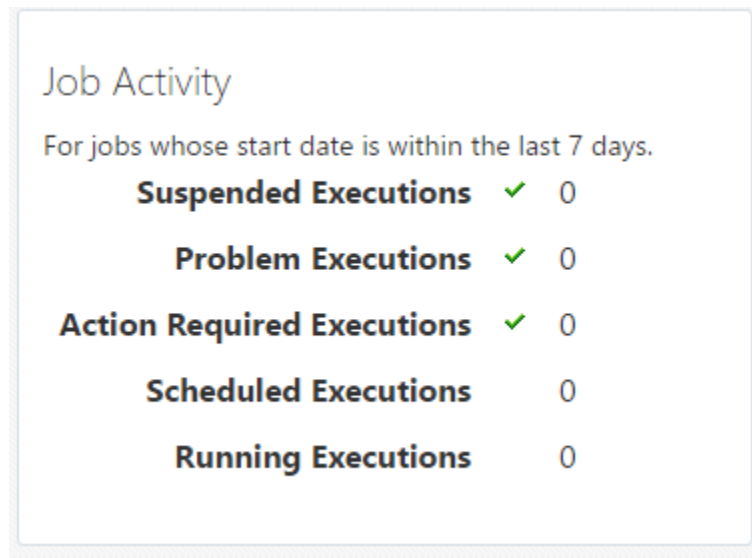


## Credentials Region

For information about reusable credentials UI components, see Reusable Credentials UI Components.

## Defining Charts

MPCUI supports the chart standard set out by JET. You can feel free to use any chart type which JET supports. MPCUI supports three chart types in particular. All of these chart types have integral support for displaying metric information by specifying the metric properties. Additionally, you can construct your own data for the chart using information obtained from other services including SQLDataService and map it to the charts using the dataService property.

The following examples and documentation for each chart type are a brief summary of the various options available for each chart. For a complete description of each chart's properties, refer to the API documentation. For examples of how these charts work at runtime, see the Demo Sample included in the Extensibility Development Kit.

### Line Chart

Typically, the line chart displays information over time (often referred to as a time-series chart). Therefore, it lends itself to the display of metric information either historically or in real-time. The chart includes properties for specifying the metricName and metricColumns (an array) that should be shown in the chart and a timePeriod property that can be set to show historical data or real-time sampled data. When timePeriod is set to "REALTIME", the chart manages an automatic polling request for you and updates the chart data as new samples arrive.

```
<mp-section id="loadChartSection" params="title: getString('CPU_LOAD')"
            style="width:75%" >
  <mp-chart id="loadChart" style="height:100%;width:100%"
            params="mpChart: {
```

```
                         type: 'line',
                         dataSelection: 'multiple',
                         emptyText: 'No data',
                         yAxis: {min: 0, max: 2},
                         styleDefaults: {colors: Colors.DEFAULT_COLORS},
                         animationOnDisplay: 'auto',
                         targetName: appModel.target.name,
                         targetType: appModel.target.type,
                         metricName: 'Response',
                         metricColumns: ['Load'],
                         timePeriod: 'LAST_DAY'}"
      ></mp-chart>
   </mp-section>
```

**Figure 9-13   Example of a Line Chart**
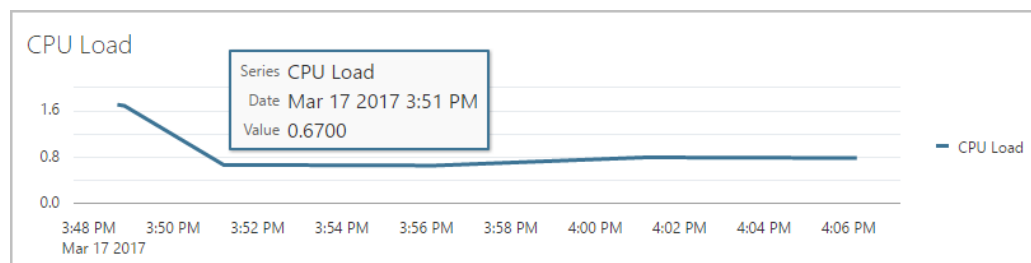


## Providing Line Chart Data Source

In addition to specifying metrics to be plotted using the line chart, you can create your own data source that is used by the chart to display data. For example, data obtained through the SQL data service or some other means such as by using the polling service and then creating the data samples to be added to the chart.

In the following example, the page includes a chart with the customDataSource mapped to an item in the page model that is constructed in the page controller.

- **ProcessesPage.html**

```
<mp-chart id="lchart_from_custom" style="height:100%;width:100%"
          params="mpChart: {
              type: 'line',
              customDataSource: model().cpuChartData,
              emptyText: 'No data',
              animationOnDisplay: 'auto'}"
 >
</mp-chart>
```

- **ProcessesPageController.js** (init method)

```
// setup a data provider for the CPU line chart; it will be
// updated each time a new data sample comes back for this metric

// first get the polling context for a 15 second interval
var pollingCtx =
          page.pollingContext.getContext(PollingInterval.EVERY_15_SECONDS);

// now get the metric to be selected and initiate the request (won't start
until
```

```
// "startPolling" is called)
var cpuPerf = TargetContext.getTargetContext().getMetric("CPUPerf");
var cpuPerfSel = cpuPerf.getSelector(['system', 'idle', 'io_wait']);
cpuPerfSel.getData(page.cb(this.cpuDataHandler),
                   MetricCollectionTimePeriod.REALTIME, pollingCtx);

// start polling; this will automatically stop when user moves to another
page
pollingCtx.startPolling();
```

- **ProcessesPageController.js** (cpuDataHandler method)

```
ProcessesPageController.prototype.cpuDataHandler = function(cpuData, fault)
{
   var page = this.page;
   if (fault != null || result.errorMessage != null) {
     MpLog.logError(fault, "Getting CPU Data via Metric Service");
     return;
   }

   var cpuData = page.getModel("cpuChartData");
   if (cpuData == null) {
     cpuData = new CustomDataSource(["Sys/IO", "Idle %"], false, true);
      page.setModel("cpuChartData", cpuData);
   }

   var cpuPt = result.results[0];
   var cpuSys = cpuPt.data[0]['system'];
   var cpuIO = cpuPt.data[0]['io_wait'];
   var cpuIdle = cpuPt.data[0]['idle'];
   var derivedCpu = cpuSys+cpuIO;

   cpuData.setTimestampedRow(cpuPt.timestamp, [derivedCpu,
cpuIdle]);   };
```

## Controlling the Legend

All charts can include a legend that displays the items shown in the chart. Use the
following example to position the legend in one of four locations (top, bottom, start,
end). The legend settings are managed as a part of the JET interface, and are
documented by the JET API documentation.

```
<mp-chart id="lchart" style="height:100%;width:100%"

     params="mpChart: {
             type: 'line',
             customDataSource: model().cpuChartData,
             emptyText: 'No data',
             legend: {position: 'bottom'},
             animationOnDisplay: 'auto'}"
     >
```

</mp-chart>

## Area Chart

The area chart is similar to the line chart and has the same attributes. It displays data
in the same way as LineChart. The showCumulativeLine property controls the display
of an area chart. For most area charts, this property should be included in set to "true"

to show a stacked or cumulative area chart. Otherwise, the area chart overlays the fill areas for each series included in the chart.

```
<mp-chart id="cpuutil" style="width:100%;height:100%"
        params="mpChart: {
                type: 'area',
                metricName: 'CPUProcessorPerf',
                metricColumns: ['CPUIdle'],
                timePeriod: 'LAST_DAY'}">
</mp-chart>
```

# Bar (Horizontal) Chart

The bar chart exposes the same properties as the column chart both for visible attributes and for specifying control over the data source:

```
<mp-chart id="spaceChart" style="width:100%;height:100%"
    params="mpChart: {
            type: 'bar',
            orientation: 'horizontal',
            timePeriod: 'CURRENT',
            groupBy: 'byKey',
            metricName: 'MSSQL_Database',
            metricColumns: ['spaceavailable']}">
</mp-chart>
```

**Figure 9-14    Bar Chart**
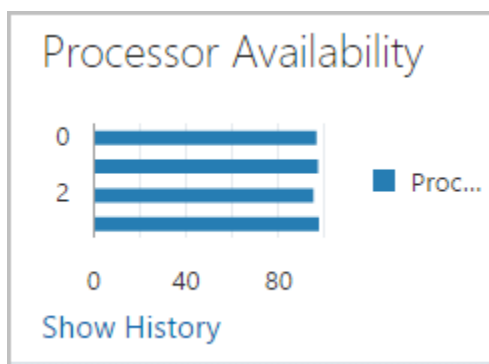


# Grouping Bars

The `groupBy` property (available for bar and column charts) enables you to organize data by key or by column. The default (by column) applies when the data set does not include keys.

In the following example, the `groupBy` property is set to **byColumn**. This creates two groups of columns, one for each data column, with all three keys appearing in each group as displayed in Figure 9-15.

**Example: Group By Column**

```
<mp-chart id="userBarChart" style="width:100%;height:100%"
    params="mpChart: {
            type: 'bar',
            orientation: 'horizontal,
```

```
            customDataSource: model().userData,
            groupBy: 'byColumn'}">
</mp-chart>
```

**Figure 9-15    Group By Column**
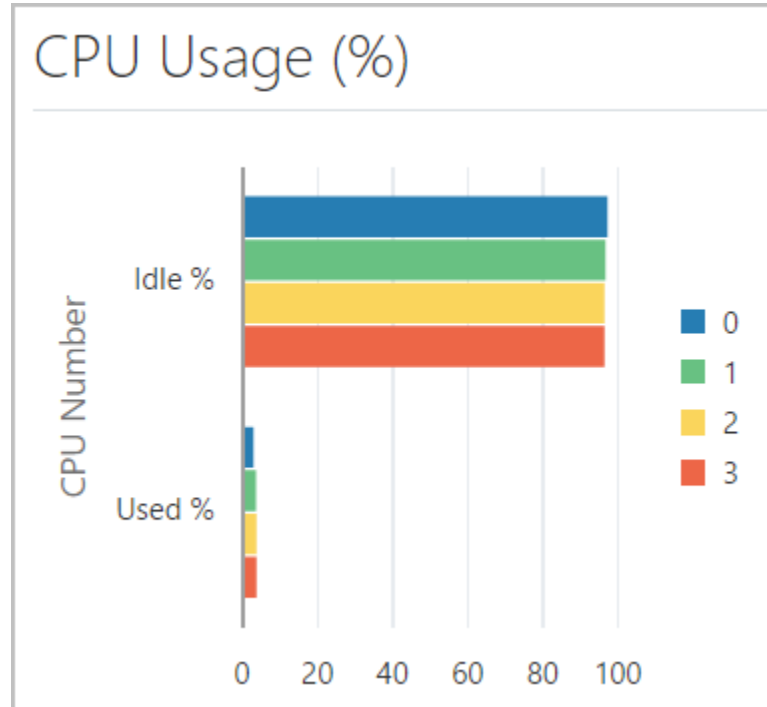


In the following example, the `groupBy` property is set to **byKey**. This creates three groups, one for each key, with both columns (the data items) appearing in each group as displayed in Figure 9-16.

**Example: Group by Key**

```
 <mp-chart id="userBarChart" style="width:100%;height:100%"
     params="mpChart: {
         type: 'bar',
         orientation: 'horizontal,
         customDataSource: model().userData,
         groupBy: 'byKey'}">
</mp-chart>
```

**Figure 9-16    Group By Key**
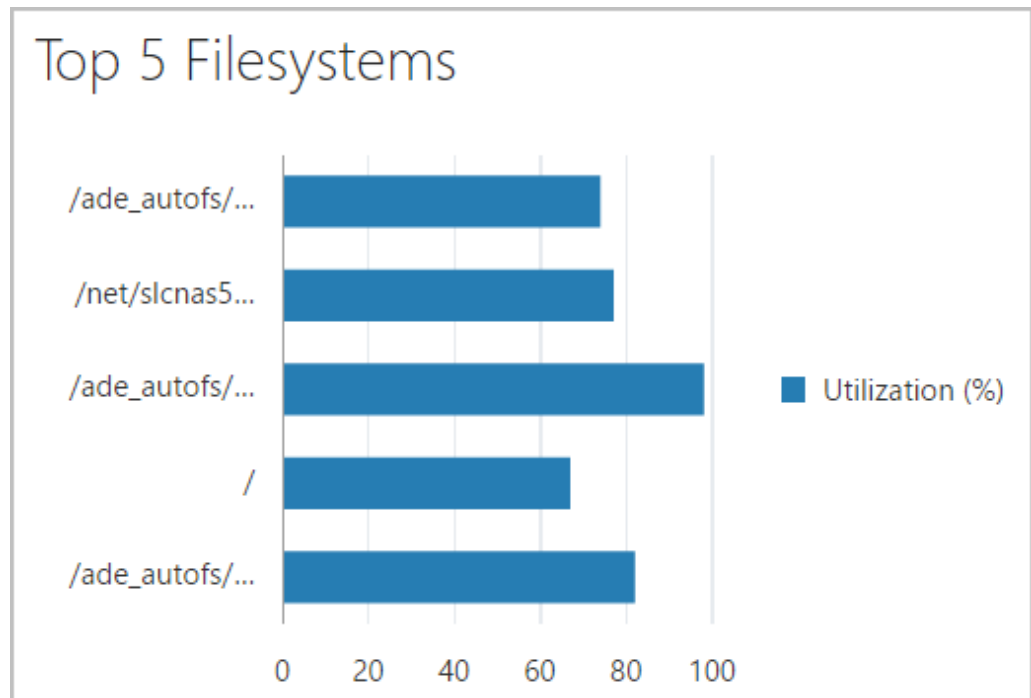


## Bar (Vertical Bar) Chart

The default setting for the bar chart is to have vertical bars. It exposes the same properties as the horizontal bar chart both for visible attributes and for specifying control over the data source:

```
<mp-chart id="brChart" style="width:100%;height:100%"
    params="mpChart: {
        type: 'bar',
        metricName: 'CPUProcessorPerf,
        metricColumns: ['CPUIdle'],
        timePeriod: 'LAST_DAY',
        groupBy: 'byKey'}">
</mp-chart>
```

**Figure 9-17    Bar Chart**
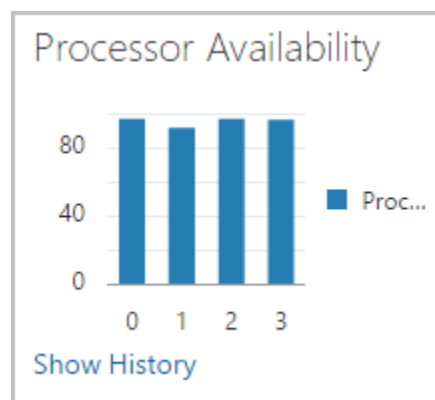
## Pie Chart

In the following example, the code constructs a pie chart by specifying the metric name and metric columns. The MPCUI framework performs the necessary requests to obtain information from the Management Server and populates the values in the chart.

> **Note:**
>
> For the `metricColumns` attribute, the value is set in the controller (see the `HomePageController.js` example) in response to the user changing the value of the combo box above the chart.

```
<mp-chart id="memChart" style="height:calc(100% - 35px);width:100%"
          params="mpChart: {
                type: 'pie',
                dataSelection: 'multiple',
                styleDefaults: {colors: Colors.DEFAULT_COLORS},
                targetName: appModel.target.name, targetType:
appModel.target.type,
                metricName: 'MemoryPerf',  metricColumns: model().memChartColumns,
                timePeriod: 'CURRENT',
                emptyText: 'No data',
                animationOnDisplay: 'auto'}"
    >
</mp-chart>
```

**Figure 9-18    Pie Chart**
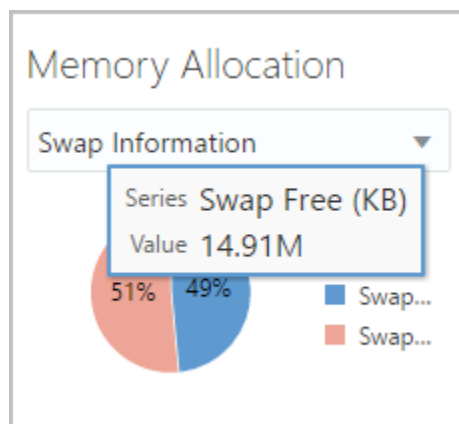


## Defining Tables

The following sections describe the different methods of defining tables, providing examples of each method.

## Data Service

The following example maps the table to the MetricDataService by specifying the metricName and metricColumns. You do not have to specify the headerText attributes for the columns because it will be filled with the metric column labels. You can override these labels if required.

**Example: Mapping a Table to the MetricDataService**

```
<mp-section id="fsrowc2" params="title : ''" style="width:65%">
  <mp-section id="fsrowc2sec" style="height:10%" params="title : ''">
     <div id='toolbar-container' class="mp-flex-item"
style="height:100%;width:100%">
       <div id="myToolbar" aria-controls="player" style='float:right'
           data-bind="ojComponent: {component:'ojToolbar',
                                     tableId: 'processesTable'}">
         <button id="killProcessButtonJ"
                 data-bind="mpTableAdminButton: {
                         label: 'Kill (Job)',
                         adminClick: cb(controller.killProcess, true, false)
                 }">
         </button>
       </div>
     </div>
  </mp-section>
  <mp-section id="fsrowc2sec2" style="height:90%" params="title : ''">
    <mp-table id="processesTable"
              params="mpTable: {
                    selectionMode: {row: 'single', column: 'multiple'},
                    metricName: 'CPUProcessesPerf',
                    metricColumns: ['ProcUser', 'ProcCPU', 'ProcCmd'],
                    timePeriod: 'REALTIME',
                    interval: 30,
                    dataUpdateListener: cb(controller.processesTableUpdated)
                    }, mpPagingControl: { pageSize: 15 }">
    </mp-table>
  </mp-section>
</mp-section>
```

> **Note:**
>
> The `mpPagingControl` accepts the same inputs as the JET API for paging. This interface folds those settings directly into the parameters for the `mp-table` tag, but you are free to use all documented JET settings here.

**Figure 9-19    Data Service**



## Custom Data Provider

In the following example, the data for the table is loaded in the controller, and mapped to the page model processInfoData item. The processInfoData is an array of objects (of any type). The field property specified for each column identifies the public property that will be displayed in each column. In this case, the fieldname will also be used as the headerText. You can supply the headerText property to override this label.

**Example: Mapping a Table to the processInfoData Item**

```
<mp-table id="processInfoTable" style="width:100%;height:100%"
          params="mpTable: {
                customDataSource: model().processInfoData,
                columns: [
                  {headerText: 'Process ID', field: 'Process ID', id: 'pid',
headerStyle: 'width:100px'},
                  {headerText: 'User', field: 'User', id: 'User', headerStyle:
'width:250px'},
                  {headerText: 'Database', field: 'Database', id: 'Database',
headerStyle: 'width:100px'},
                  {headerText: 'Status', field: 'Status', id: 'Status',
headerStyle: 'width:100px'},
                  {headerText: 'Command', field: 'Command', id: 'Command',
headerStyle: 'width:250px'},
                  {headerText: 'CPU Time', field: 'CPU Time', id: 'CPUTime',
headerStyle: 'width:100px'},
                  {headerText: 'Memory Usage', field: 'Memory Usage', id:
'MemoryUsage', headerStyle: 'width:100px'}
                ]}"
                >
</mp-table>
```

## Getting Selected Rows

The rows currently selected in the table can be obtained from the selection property of the JET table. This property is documented in the JET API documentation, and it contains only start and end index values; no actual data. Those index values will have to be mapped back to the underlying data object, either by retrieving it from the table object or from the page model.

```
 // Get the child node in the mp-table
var tableId=$
("#processesTable").data(CustomElement.MPCUI_CHILD_NODE).getChildNodeId();
// Get the get the selection object from the JET table (child node)
 var selection = $("#" + tableId).ojTable("option", "selection");
 var selectionRow = (selection == null) ? null : Util.def(selection[0], null);
 if (selectionRow == null) {
   MessageAlert.show("No process has been selected.  Select the process to stop
from the table below.", "No Process Selected");
   return;
 }
//  The selection object is documented in the JET API reference, but it contains
no
//  data, only index references into the underlying data object.
 var selectionRowIdx = selectionRow.startIndex.row;  // Single row,
startIndex=endIndex
// Access the underlying data object
 var rawTableData = table.ojTable("option", "data");
 var processPromise = rawTableData.at(selectionRowIdx);
 processPromise.then(Util.createProxy(this, this.confirmPromiseHandler));
```

# Defining Dialogs

When you construct a dialog, typically you require an HTML file only, using the mp-dialog tag to start the HTML content.

# Dialog Registration

To make a dialog available to be displayed using the `invokeActivity` method, you must register it as an activity as part of the Integration class. In the following example, note the following:

- `id` attribute: The ID is used to reference this dialog from other activities within the application. It must be unique across all activities included in the application.

- `dialogClass` attribute: The `dialogClass` attribute is a reference to the HTML which is the implementation for this dialog.

`inputParams` are optional, but they enable the dialog to be reused in situations where input parameters are required and you want to pass an object as context directly from the HTML using the bean directive. The MPCUI framework maps the input object parameters to the dialog parameters.

If you do not define `inputParams` as part of the dialog definition, then any input data required by the dialog (such as any custom properties) would have to be set in JavaScript and the dialog shown using the `Dialog.show` method.

**Example: Registering a Dialog**

```
<mp:DialogActivityDef id='metricHistory' label='Metric History'
dialogClass='dhs/MetricHistoryDialog' >
          <mp:inputParams>
                <mp:InputParam name='targetName'/>
                <mp:InputParam name='targetType'/>
                <mp:InputParam name='metric'/>
                <mp:InputParam name='columns'/>
                <mp:InputParam name='period'/>
                <mp:InputParam name='title'/>
```

```
            </mp:inputParams>
         </mp:DialogActivityDef>
```

**Figure 9-20    Metric History Dialog**
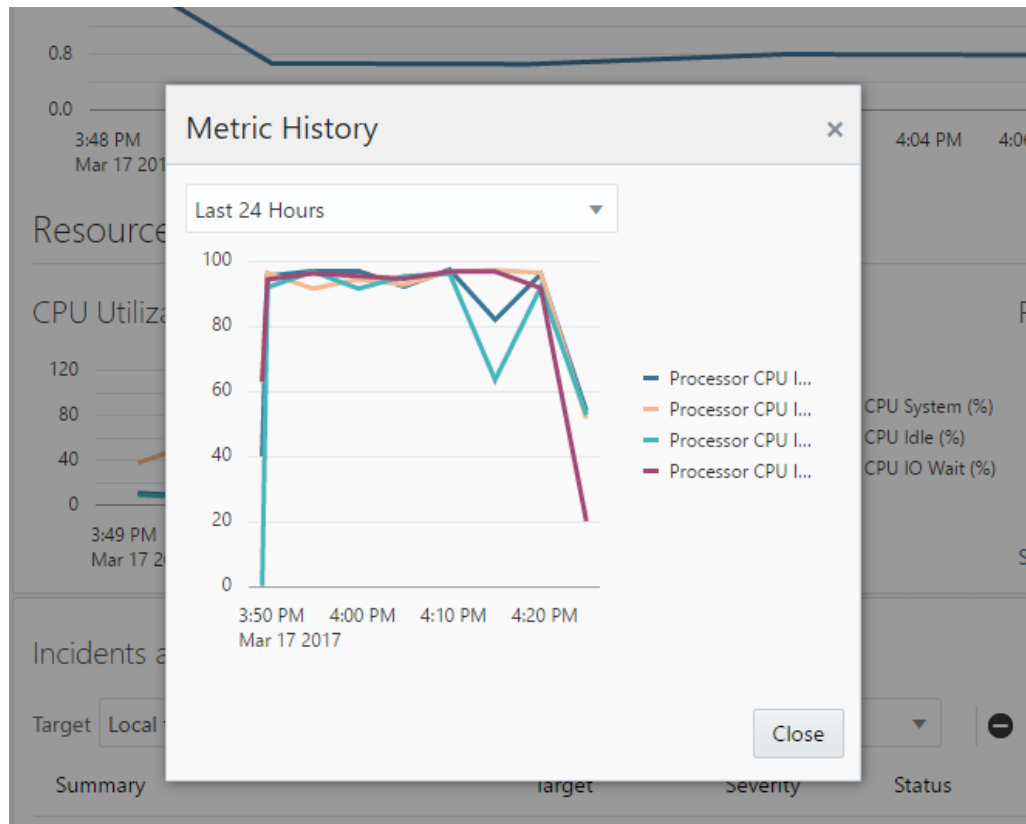


## Displaying a Dialog and Waiting for Close Events

If the dialog includes some state that is required when the dialog closes, then a close handler can be passed to the invokeActivity method. This handler is called with the CloseEvent. This handler identifies which button was pressed to close the dialog and retrieves the dialog object itself to retrieve information from it.

**Example: Waiting for a Close Event**

```
HomePageController.prototype.showRpmDialog = function(data, event) {
  this.page.invokeActivity('rpmInfoDialog', null,
this.page.cb(this.rpmInfoDone));
};
HomePageController.prototype.rpmInfoDone = function(event) {
  ...
};
```

In the previous example, the rpmInfoDone function is passed to invokeActivity. When the dialog closes, the method is called and passed a CloseEvent.

# Defining Trains

The train allows the definition of a multi-step UI, with next and previous buttons to navigate between each step. The train is typically used in cases where the user is going to create or modify an entity that has a large number of attributes that can be organized into categories.

The train must be registered with the integration metadata, and includes a controller that extends TrainController and a page for each step in the train and uses the mp-train-step-page tag. Each step (train step page) can have its own controller class. Because each step is a page with a controller, the layout, management of data and response to events within the step is exactly the same as any other page in the application. For more information about the Page, see Page.

The train step controller can access the train itself by referencing the TrainStepPage.train property. Use this to access other information maintained within the train object or its model.

## Train Definition Example

The following example provides a definition of train and the next example shows the train.

**Example: Defining a Train**

```
<mp:TrainActivityDef id='addNewUserEmbeddedTrain' label='Add New User'>
  <mp:stepActivities>
    <mp:TrainStepActivityDef id='anuStep1' label='User Info' pageClass='dhs/user/
UserInfo' pageControllerClass='dhs/user/AddNewUserTrainStepController'/>
    <mp:TrainStepActivityDef id='anuStep3' label='Credentials' pageClass='dhs/
user/Credentials' pageControllerClass='dhs/user/AddNewUserTrainStepController'/>
    <mp:TrainStepActivityDef id='anuStep2' label='Expiry' pageClass='dhs/user/
Expiry' pageControllerClass='dhs/user/AddNewUserTrainStepController'/>
    <mp:TrainStepActivityDef id='anuStep4' label='Schedule' pageClass='dhs/user/
Schedule' pageControllerClass='dhs/user/AddNewUserTrainStepController'/>
    <mp:TrainStepActivityDef id='anuStep5'
label='Notifications' pageClass='dhs/user/Notifications'
pageControllerClass='dhs/user/NotificationsTrainStepController'/>
    <mp:TrainStepActivityDef id='anuStep6'
label='Confirmation' pageClass='dhs/user/Confirm' pageControllerClass='dhs/user/
AddNewUserTrainStepController'/>
  </mp:stepActivities>
</mp:TrainActivityDef>
```

**Figure 9-21    Train Example**



## Train Controller

The train controller is used to managed state kept across all pages in the train and respond to changes in the train (movement between steps) and respond to the train completing when the user clicks either the Finish or Cancel button.

## Train State

State may be maintained in the Train model using the Train.model property. This property is a dynamic property that can be used to hold any information appropriate to the train. Individual pages can store their own state in their own model properties and may also access information stored in the train model.

## Train Events

Each train step controller can implement the init and destroy methods that are called when the step starts or stops. The step can do either of the following:

• Perform a step-specific processing step

• Access the train and allow it to process higher level logic

The train controller can also be called when the train ends (Finish or Cancel) by adding a listener function for the train done event:

**Example: Adding a Listener Function**

```
 // register a listener for the train complete event, this may be a cancel or
finish.
 train.addEventListener(TrainEvent.TRAIN_EVENT, trainDone);
```

The listener (`trainDone` in the previous example) can inspect the train state and determine if processing should continue or not. It can choose to direct control to some other activity (page) or can set the train back to another step:

**Example: Defining Actions at the End of the Train**

```
MyTrainController.prototype.trainDone = function(event) {
  // train cancel/finish button was pressed, so caller can now validate
  // the train (look at the model).  The caller has the various options
indicated below.
  var train = event.train;

  if(train.getModel("isComplete")) {
    // want to end the train, but go somewhere else (otherPage is a page id)
    train.endTrain("otherPage");
  } else {
    // go back to train at a certain step
    train.controller.setStepById("step2");
  }
};
```

# Defining Information Item and Information Displays (Label-Value Pairs)

The `InfoDisplay` and `InfoItem` classes allow you to display a set of label-value pairs in a group with the labels right-aligned and the values left-aligned. Each entry (`InfoItem`) in the display specifies a label, value, optional icon, destination, or click property.

The destination or click properties cause the value to appear as a link. You can set destination to either of the following:

- String that is the identifier for some other activity (page or dialog)

- URL object constructed in the controller
  (see `HomePage.html` and `HomePageController.js` for examples)

You can specify the click handler instead of the destination and set it to a function within the controller that will be called when the item is clicked by the user.

**Example: Defining Label Value Pairs**

```
<mp-info-display>
  <mp-info-item id="cpuModel" params="label : getString('CPU_MODEL'),
                               value: model().configData().cpuModel"></mp-info-
item>
  <mp-info-item id="cpuIdle" params="label : getString('CPU0_IDLE'),
                              value: procData.result.getString('0','CPUIdle'),
                              imageRenderer: rendererFactory.get('CHECK_MARK',
bean('type','number','warning','95','critical','99'))
     "></mp-info-item>
  <mp-info-item id="osVersion" params="label : getString('OS_VERSION'),
                                value: model().configData().osVersion"></mp-
info-item>
  <mp-info-item id="hostedBy" params="label : getString('HOSTED_BY'),
                               value: model().host().info().name "></mp-info-
item>
</mp-info-display>
```

**Figure 9-22    Label Value Pairs**



# Using Built-in Renderers

In addition to the ability to define custom renderers for table columns, headers, and other UI elements using the capabilities provided by the JET framework, the MPCUI framework also provides several built-in renderers that can be used to display custom icons in a table or for an InfoItem.

These built-in renderers show an icon in place of a text value, either in a Table or InfoItem component. The renderer is specified by using the "rendererFactory" directly in the HTML and specifies a renderer id to select the renderer and then a set of input parameters for the renderer depending on the renderer type.

For example, the following code results in an icon being displayed next to the value on the InfoItem and shows a check mark, warning, or error icon depending on the value displayed in the InfoItem:

```
<mp-info-item id="cpuIdle" params="label: getString('CPU0_IDLE'),
                          value: procData.result.getString('0','CPUIdle'),
                          imageRenderer: rendererFactory.get('CHECK_MARK',

bean('type','number','warning','95','critical','99'))
    "></mp-info-item>
```

The first parameter, "CHECK_MARK" indicates which renderer to be used (see the complete list below). The second parameter, "bean" specifies the input parameter for the check mark renderer. This parameter will be different and in some cases optional depending on the renderer selected. Refer to the API documentation for details regarding what each renderer requires for input parameters.

The built-in renderers include the following:

*   CHECK_MARK

    Displays a check mark, warning icon or error icon depending on the value provided. The renderer can either be used to display a check mark or error icon in the case where a Boolean value is shown. The Boolean may be true/false, t/f or 0/1. If the 'type' parameter is specified as 'number', then the value will be compared to thresholds provided in the input parameters to also show a different icon if the following is beyond the specified threshold.

- TARGET_TYPE

    Displays the icon associated with a target type value. This is the internal
    target type id, such as 'oracle_database', and not the actual displayed string
    representation of the target type.

- TARGET_STATUS

    Displays the icon associated with a status value. The status value will be up/down,
    true/false, or 0/1 and will display an up or down arrow according to that value.

When the renderer is associated with an InfoItem, by default the value shown in the
InfoItem will be passed to the renderer to determine which icon should be displayed. In
cases where an alternative value should be used to control the renderer, the InfoItem
provides the "imageDataSource" property. This property can be bound to a data item
that is different than the displayed value.

# Defining Links

Use the link component to display what appears to the user to be a link to a URL. The
link specifies a label property and also either a destination or click handler property.
The destination can be an activity id or a URL object constructed in the controller.  For
information about the `InfoItem` class, see Defining Information Item and Information
Displays (Label-Value Pairs).

**Figure 9-23    Link Example**
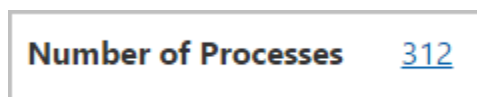


# Including Enterprise Manager Images

To reference one of the images shipped with the Enterprise Manager product from the
HTML, use the appModel.emImage function to refer to the desired image. Note that
the list of images and their filenames is not currently part of the Enterprise Manager
EDK and therefore is subject to change.

You should verify and test any use of this information with each new release of
Enterprise Manager. In a typical deployment, the images are located under the
emcore_war/images directory. For example::

```
<img data-bind="attr: {src: appModel.emImage('yellowFlag.gif')}" />
```

# Displaying a Processing Cursor

The UI displays the processing or busy cursor automatically when:

- Any new activity is accessed (page, train or dialog)

- Any request is made to the Management Server for data

Typically, you do not have to show the busy cursor. However, if you feel that you
must show the busy cursor, take care that the cursor is ended cleanly. Ensure that if

exceptions are thrown while the busy cursor is shown, that they are caught and the cursor is removed.

To show the cursor, call the `MpCursor.setBusyCursor` method.

To remove the cursor, call `MpCursor.removeBusyCursor`. Both methods accept an optional `owner` parameter. This parameter allows you to nest multiple cursors calls.

# Defining Icons for Target Types

You can specify icons to associate with a target type to be displayed in the Cloud Control console wherever a target type icon is shown (such as next to the target menu).

MPCUI supports the following graphic formats for icons:

*   PNG

*   JPG

*   GIF

Oracle recommends the following sizing for icons:

*   Small icon: 16x16

*   Large icon: 24x24

Save the icon files in the *plugin_stage*/oms/metadata/mpcui directory. For more information about the plug-in staging directory, see Staging the Plug-in.

**Example: Defining Icons**

```
<EmuiConfig>
    <large-icon>demo_hs_large_icon.png</large-icon>
    <small-icon>demo_hs_small_icon.png</small-icon>
</EmuiConfig>
```

Figure 9-24 and Figure 9-25 provide examples of a small and a large icon.

**Figure 9-24    Small Icon**



**Figure 9-25    Large Icon**



# Displaying the Target Navigator

The target navigator can be displayed on the left side of the home page of any composite target or any of its members. The target navigator displays the composite target at its root and then shows all members of the composite by searching for any *contains* associations below it. Targets that are associated with the composite target can have other *non-contains* associations with the composite target or with other

targets. However, only those targets with *contains* associations with the composite target are shown in the target navigator. You can add these containment associations through any of the supported mechanisms for discovering or deriving associations. For more information, see Using Derived Associations.

To enable the target navigator, the MPCUI metadata must include the `<EmuiConfig>` element with the `context-pane-visible` property set to true. This must be set for the composite target type as well as any of its member targets. If it is not set for member targets, then the navigator will not appear showing the other members of the composite when the home page is displayed for those targets.

By default, the `context-pane-visible` property is set to false and the target navigator is not displayed.

> **✏ Note:**
>
> If there are no *contains* associations, then the target navigator will not appear, even if the `context-pane-visible` property is set to true.

**Example: Enabling the Target Navigator**

```
<?xml version = '1.0' encoding = 'UTF-8'?>

<CustomUI target_type="demo_hostsystem"xmlns="http://www.oracle.com/
EnterpriseGridControl/MpCui">

  <EmuiConfig>
     <context-pane-visible>true</context-pane-visible>
  </EmuiConfig>

</CustomUI>
```

# Defining a UI for Guided Discovery

The MPCUI framework supports the ability to define a custom user interface that can be registered as part of a guided discovery flow. After registration, this discovery flow is available from the **Add Targets Manually** page. For more information about adding targets manually, see Manually Adding Targets .

## About Guided Discovery

The guided discovery flow provides you with the ability to add targets and associations to Enterprise Manager by running discovery scripts on selected Management Agents and calling service APIs to add the appropriate entities. This process is driven from a user interface wizard (train) and can use information supplied by the end user to guide the process. It is up to you to determine (based on your specific requirements) the information required from the end user during this process. For examples, see the plug-in samples in the EDK (demo_hostsample and demo_hostsystem). For more information about discovery scripts, see Defining Target Discovery .

The services typically used during guided discovery include the following:

- TargetInfo services to retrieve Management Agents and targets, for example, for target existence or target properties

- AssociationInfo services to retrieve existing associations

- Discovery service to run discovery scripts on selected Management Agents

- TargetManagement services to create or delete targets

- AssociationManagement services to create or delete associations

For more information about these services, see Using Discovery Service , Using Target Information Services, and Using Target Management Services.

## Supporting Guided Discovery

To add guided discovery to a plug-in, ensure that the following directories contain the required files:

- *plugin_stage*/discovery

  - Scripts that will be executed from the guided discovery flow. These scripts might include multiple targets and associations. For more information about discovery scripts, see Creating the Discovery Script.

  - customdiscover.lst file. This file must include one line for each discovery script to be provided with the plug-in. Each entry must reference a discovery category, which is a unique identifier that will be used to identify the script to be executed when calling the discovery service. The following entry shows a discovery category (DHS_DISC) that is used to refer to the demo_hostsample_discovery.pl script during the guided discovery flow.

    ```
    DHS_DISC|demo_hostsample_discovery.pl
    ```

- *plugin_stage*/oms/metadata/discovery

  Discovery metadata file (*plugin*_discovery.xml). For more information about the discovery metadata file and an example of the discovery XML, see Creating Discovery XML. For guided discovery, there are a number of attributes that must be specified correctly to allow your guided discovery to be registered correctly.

  - ```
    <DiscoveryModule name="DemoHostSample">
    ```

    This is the unique name for the discovery module and must match the module name used to register the discovery SWF in the MPCUI metadata file.

  - ```
    <NlsValue>Discover Demo Host Sample Targets</NlsValue>
    ```

    This is the label that appears in the Target Types list on the **Add Targets Manually** page of the Cloud Control console.

    ```
    <CustomDiscoveryUI>
        <LaunchADF>
           <DestOutcome>goto_core-mpcustomdiscovery-page</DestOutcome>
        </LaunchADF>
    </CustomDiscoveryUI>
    ```

    This must be exactly the same in your discovery metadata file. It ensures that the guided discovery UI that you built and included in your plug-in will be launched.

- *plugin_stage*/oms/metadata/mpcui

  - *discovery*.js

    Similar to the MPCUI JS library created for a target home page, the guided discovery UI is constructed as a JS library.

– *MyMpcui*.xml

In addition to the discovery JS library, the MPCUI metadata file must include Integration metadata for the discovery module.

**Example: SwfFiles Tag From MPCUI Metadata File**

```
<mp:Integration mpcuiLibVersion="13.2.0.0.0" integrationType="discovery"
                discoveryModule="DemoHostSample"
      xmlns:mp="http://www.oracle.com/EnterpriseGridControl/
MpCuiIntegration"
>
…
</mp:Integration>
```

The `<mp:Integration>` entry is similar to the one created for the target instance UI. However, this one specifies the `discoveryModule`. This UI is launched when this discovery module is selected by the user in the **Add Targets Manually** page in the Cloud Control console.

# Constructing the Guided Discovery User Interface

The guided discovery UI is built using the MPCUI features for constructing an HTML/JS UI. The UI components, such as regions, buttons, tables, dialogs, and so on are used to construct a user interface to guide the user through the process of adding new targets to Enterprise Manager. For information about adding these UI components, see the relevant sections of this chapter, such as Defining Tables or Defining Dialogs.

# Discovery Integration

The integration metadata for the discovery UI defines the set of activities used by the discovery UI. The discovery UI must include at least one `PageActivity` that is defined with the `isDefaultPage=true` property indicating that this is the page that will be loaded when the guided discovery starts. In the following example, (extracted from the demo_hostsystem sample plug-in), take note of the `discoHomePg` activity.

**Example:  Integration Metadata**

```
<mp:Integration mpcuiLibVersion="13.2.0.0.0" integrationType="discovery"
                discoveryModule="DemoHostSample"
                xmlns:mp="http://www.oracle.com/EnterpriseGridControl/
MpCuiIntegration"
            >
  <mp:sourceContext>
    <mp:jsRoot path="js"/>
    <mp:bundleRoot path="rsc"/>
  </mp:sourceContext>

  <mp:jsLibraries>
    <mp:jsLibrary id="pluginLib" path="libs/dhs/demo_hostsample_discovery-
min.js"
                  debugPath="libs/dhs/demo_hostsample_discovery-debug.js"
                  version="13.2.0.0.0" isDefault="true">
      <mp:jsModule module="ojs/ojmodel"></mp:jsModule>
      <mp:jsModule module="ojs/ojknockout"></mp:jsModule>
      <mp:jsModule module="ojs/ojknockout-model"></mp:jsModule>
      <mp:jsModule module="ojs/ojcomponents"></mp:jsModule>
      <mp:jsModule module="ojs/ojarraytabledatasource"></mp:jsModule>
```

```
            <mp:jsModule module="ojs/ojdatetimepicker"></mp:jsModule>
            <mp:jsModule module="ojs/ojtable"></mp:jsModule>
            <mp:jsModule module="ojs/ojdatagrid"></mp:jsModule>
            <mp:jsModule module="ojs/ojchart"></mp:jsModule>
            <mp:jsModule module="ojs/ojgauge"></mp:jsModule>
            <mp:jsModule module="ojs/ojlegend"></mp:jsModule>
            <mp:jsModule module="ojs/ojselectcombobox"></mp:jsModule>
            <mp:jsModule module="ojs/ojsunburst"></mp:jsModule>
            <mp:jsModule module="ojs/ojthematicmap"></mp:jsModule>
            <mp:jsModule module="ojs/ojtreemap"></mp:jsModule>
            <mp:jsModule module="ojs/ojvalidation"></mp:jsModule>
            <mp:jsModule module="ojs/ojslider"></mp:jsModule>
        </mp:jsLibrary>
    </mp:jsLibraries>
    <mp:resourceBundles>
        <mp:MpBundle name="demoUiMsg" path="oracle.samples.xohs.rsc"
isDefault="true"/>
        <mp:MpBundle name="demoJobMsg" path="oracle.samples.xohs.rsc"/>
    </mp:resourceBundles>

    <mp:activities>
        <mp:PageActivityDef id='discoHomePg' label='Discovery Console'
                            pageClass='dhs/discovery/DiscoveryTrainPage'
                            pageControllerClass='dhs/discovery/
DiscoveryTrainPageController'
                            isDefaultPage="true" />
        <mp:TrainActivityDef id='discoTrain' label='Discover New Targets'
                            trainControllerClass='dhs/discovery/
DiscoveryTrainController'>
            <mp:stepActivities>
                <mp:TrainStepActivityDef id='selAgentsStep' shortLabel="Select Agents"
                                         label='Add Demo Host Sample Targets: Select
Agents'
                                         pageClass='dhs/discovery/SelectAgentStep'
                            pageControllerClass='dhs/discovery/
DiscoveryStepController'/>
                <mp:TrainStepActivityDef id='agentInputStep' shortLabel="Configure
Inputs"
                                         label='Add Demo Host Sample Targets: Configure
Inputs'
                                         pageClass='dhs/discovery/AgentParamInputStep'
                            pageControllerClass='dhs/discovery/
DiscoveryStepController'/>
                <mp:TrainStepActivityDef id='selTargetsStep' shortLabel="Configure
Targets"
                                          label='Add Demo Host Sample Targets: Configure
Targets'
                                          pageClass='dhs/discovery/
SelectTargetsStep'
                            pageControllerClass='dhs/discovery/
DiscoveryStepController'/>
                <mp:TrainStepActivityDef id='summaryStep' shortLabel="Summary"
                                          label='Add Demo Host Sample Targets: Summary'
                                          pageClass='dhs/discovery/FinalizeStep'
                            pageControllerClass='dhs/discovery/
DiscoveryStepController'/>
            </mp:stepActivities>
        </mp:TrainActivityDef>
        <mp:DialogActivityDef id='configureInstancePropertiesDialog'
                            label='Dialog InstProp'
                            dialogClass='dhs/discovery/
```

```
ConfigureInstancePropertiesDialog'
      dialogControllerClass="dhs/discovery/
ConfigureInstancePropertiesDialogController">
      <mp:inputParams>
        <mp:InputParam name='properties'/>
        <mp:InputParam name='discoveredName'/>
        <mp:InputParam name='rowIndex'/>
      </mp:inputParams>
    </mp:DialogActivityDef>
    <mp:DialogActivityDef id='targetHomeDialog' label='Dialog TargetHome'
                          dialogClass='dhs/discovery/TargetHomeDialog'
                      dialogControllerClass="dhs/discovery/
TargetHomeDialogController">
      <mp:inputParams>
        <mp:InputParam name='targetHome'/>
        <mp:InputParam name='rowIndex'/>
      </mp:inputParams>
    </mp:DialogActivityDef>
  </mp:activities>
</mp:Integration>
```

# Structure of the Discovery UI

The discovery UI is often a single page that either has a train embedded in it, or that displays dialogs to obtain information from the end user to guide the discovery process. The steps of the guided discovery flow depends on the requirements, but often involve the following:

1. Determine on which Management Agents to run a discovery script

2. Run the discovery script

3. Process the results of the discovery script, adding additional information provided by the end user

4. Call APIs to add or delete targets

One important consideration about guided discovery is that it can be used to update the topology of existing composite targets as well as discover new targets. In the case of the sample plug-in (demo_hostsystem), the guided discovery UI allows the user to add new system targets, but also allows the user to add or remove members from an existing system.

This use case also illustrates the requirement to use Enterprise Manager APIs to query for the set of existing targets known to Enterprise Manager to compare the set with information returned from the discovery script to identify which targets are already managed by Enterprise Manager and which are not. For example, the result might be a list of new targets that should be added and a list of other targets that no longer exist in the managed configuration and must be removed from Enterprise Manager.

This scenario also illustrates that the discovery application might also be integrated with the custom UI built for the target home page. This provides the user with the ability to update the configuration of an existing composite target directly from the composite target home page.

See the **HostSystemConfiguration** page in the demo_hostsystem sample plug-in for an example of using discovery UI from within a target home page.

# Using Discovery Service

The Discovery service is used to run a discovery script included with your plug-in. For a description of your plug-in requirements to support discovery, see Supporting Guided Discovery.

The following example (included in the demo_hostsystem sample plug-in in the `DiscoveryTrainStepController`) shows calling the discovery service (`TargetFactory.discoverTargets`). This service includes an `addRequest` method that can be called multiple times to process discovery on multiple Management Agents if required.

Each call to the `addRequest` method is passed the following along with the handler that will be called with the results of the discovery script:

- Request ID

  A unique identifier (assigned by you) associated with that particular request which will enable you to retrieve the specific results associated with that request.

- Agent

  the Management Agent Target that the discovery should be run against

- Plug-in ID

  The plug-in ID associated with the discovery to be run. A plug-in can include multiple discovery modules and categories.

- Discovery category

  The discovery category. This must map to a discovery script through an entry in the discover.lst file included in the agent part of the plug-in.

**Example: Discovery Service**

```
/**
 * when doing discovery, the service will accept multiple requests to be
 * processed at the same time.  this would typically be the case if multiple
 * agents were involved in the process, but could also be if different discovery
 * categories (scripts) were to be processed.
 *
 * the discovery request includes the following elements:
 *      requestId   a unique identifier associated with that particular request
 *                  that will allow you to retrieve the specific results
associated
 *                  with that request.
 *      agent       the agent Target that the discovery should be run against
 *      pluginId    the pluginId associated with the discovery to be run; a plug-
in
 *                  can include multiple discovery modules and categories
 *      discCat     the discovery category; this must map to a discovery script
via
 *                  an entry in the discover.lst file included in the agent part
of
 *                  the plugin
 *      params      parameters that are to be passed to the discovery script
 *
 * Note on discoveryModule - in addition to the pluginId, the discovery UI is
 * passed the discovery module associated with this discovery pass.  If you've
 * chosen to implement multiple types of discovery operations from a single
```

```
 * discovery UI you may retrieve the discoveryModule to determine in what context
 * the UI was launched.
 */

var requestId = "DiscReq1";
var pluginId = ApplicationContext.getPluginId();
var discoveryCategory = "DHSYSTEM_DISC";
discSvc.addRequest(requestId, agent, pluginId, discoveryCategory, params);

var mrs = new
MultiServiceResponder(this.page.cb(this.discoverResultsHandlerMul));
var batch = this.page.getBatchRequest();

var discRequest = TargetFactory.discoverTargets(discSvc, mrs.sync, batch);
discRequest.data["initProcessing"] = initProcessing;
mrs.addRequest("discoverTargets", discRequest);

batch.sendRequest();
```

The discovery results handler, declared as follows, is passed a fault object and the discovery results.

```
DiscoveryStepController.prototype.discoverResultsHandlerMul = function(response)
{
```

If a fault did not occur during processing of the discovery script, then the response.getFault("discoverTargets") will be null. The discovery object, retrieved by response.getResult("discoverTargets"), includes an Array of the DiscoveryRequest objects constructed by calling the `addRequest` method. Each request includes the properties specified (such as agent, category, and so on) and also includes a DiscoveredTargets object. The  DiscoveredTargets object includes the list of targets returned from the discovery script that was run on the target Management Agent for the specified request. For more information about discovery scripts,  see Creating the Discovery Script and for information about the discovery objects returned by the DiscoveryService, see the API documentation in the EDK.

# Using Target Information Services

During the discovery process it is often necessary to obtain target information such as a list of agents or the set of targets of a particular type. The target information service provides a number of APIs that can be used for such purposes. This section provides an overview of these services. For additional information, see the API documentation in the EDK and for examples of their use, see the demo_hostsystem sample plug-in.

- TargetFactory.getAgents

  The getAgents API enables you to retrieve a set of Management Agents that can be used to perform discovery. You can filter the list by specifying selection properties (Array of TargetProperty) such as selecting all the Management Agents running on a Windows host.

- TargetFactory.getTargets

  Use the getTargets API to retrieve a list of targets specifying any number of selection criteria including hosts, target types, managed status, or metadata version. Each item is specified as a list of possible values and the request can include one or more selection criteria.

- Target.getSystemMembers

Use the getSystemMembers API to retrieve the list of system member targets. These are targets that are included in the system through the systemStencil. For information about the system targets, see the *Oracle Enterprise Manager Cloud Control Extensibility Programmer's Guide*.

- Target.getCompositeMembers

Use the getCompositeMembers API to retrieve the list of composite member targets. Composite members are those included in a composite (or system target) through containment associations. For information about composite targets, see the *Oracle Enterprise Manager Cloud Control Extensibility Programmer's Guide*.

## Using Target Management Services

The target management services provide you with the ability to create or delete targets or associations. In the case of target management, associations can also be passed as part of the target definition and the associations are added as part of the process of adding the target itself. This section provides an overview of these services. For additional information, see the API documentation in the EDK and for examples of their use, see the demo_hostsystem sample plug-in.

- TargetFactory.createTargets

Use the createTargets API add targets to Enterprise Manager. The process of adding targets to Enterprise Manager forces the deployment of the necessary plug-in to the Management Agents associated with each target. The request to this API is a list of Target objects, each of which must, at a minimum, specify a name, type, and agent. Typically, target instance properties (if used for this target type) can also be specified.

- TargetFactory.deleteTargets

Use the deleteTargets API to remove targets from Enterprise Manager. The request to this API is a list of Target objects. Removing a target from Enterprise Manager should be done with care as deleting the target is not reversible and it removes all target, metric, and configuration history.

- Target.createAssociations

Use the createAssociations API to add associations between the specified target and another target. Associations can be created in this way when creating them by using derived associations or by using the system stencil. In all cases, the association must be associated with a corresponding allowed pairs definition. For more information, see Using Derived Associations.

- Target.deleteAssociations

Use the deleteAssociations API to delete associations between the specified target and other targets.

## Building the MPCUI Application into a JS Library

Your MPCUI Application will be delivered and installed as a JavaScript library. All of the HTML files and JS files will be combined into a single file and minified. This ensures that your custom UI will be as peformant as possible.

## Creating the JS Library

The JS library is created with a combination of NodeJs and the RequireJS r.js script. r.js does the work of creating and minifying your library and NodeJs is used as a platform to allow you to run r.js from the command line.

1. Download and Install Node.js:

   https://nodejs.org/en/download/

2. There are build.js and build-min.js files in the tools directory. These files contain the instructions to build the debug and minified JS libraries respectively. Edit those files to list out each of your JS files and HTML files in the include statement.

3. Expand build.xml. Right click the build target and select Run Target. This target will create both the debug and minified JS library for your MPCUI Applications

4. Find the libraries in the tools/build directory.

## Adding the JS Library to The Plug-in

With the JS library built, you need to add it to the stage directory and add a reference to it in the Integration Metadata.

1. Move the library to the stage/oms/metadata/mpcui directory.

2. Add a reference to the Integration Metadata:

```
<mp:Integration mpcuiLibVersion="13.2.0.0.0"
      xmlns:mp="http://www.oracle.com/EnterpriseGridControl/
MpCuiIntegration"
  >
    <mp:sourceContext>

      <mp:jsRoot path="js/libs/dhs"/>
    </mp:sourceContext>
    <mp:jsLibraries>

      <mp:jsLibrary id="pluginLib" path="demo_hostsample-min.js"
                    debugPath="demo_hostsample-debug.js"
                    version="13.2.0.0.0" isDefault="true">
        <mp:jsModule module="ojs/ojmodel"></mp:jsModule>
        <mp:jsModule module="ojs/ojknockout"></mp:jsModule>
        <mp:jsModule module="ojs/ojknockout-model"></mp:jsModule>
        <mp:jsModule module="ojs/ojcomponents"></mp:jsModule>
        <mp:jsModule module="ojs/ojarraytabledatasource"></
mp:jsModule>
        <mp:jsModule module="ojs/ojdatetimepicker"></mp:jsModule>
        <mp:jsModule module="ojs/ojtable"></mp:jsModule>
        <mp:jsModule module="ojs/ojdatagrid"></mp:jsModule>
        <mp:jsModule module="ojs/ojchart"></mp:jsModule>
        <mp:jsModule module="ojs/ojgauge"></mp:jsModule>
        <mp:jsModule module="ojs/ojlegend"></mp:jsModule>
        <mp:jsModule module="ojs/ojselectcombobox"></mp:jsModule>
        <mp:jsModule module="ojs/ojsunburst"></mp:jsModule>
        <mp:jsModule module="ojs/ojthematicmap"></mp:jsModule>
```

```
        <mp:jsModule module="ojs/ojtreemap"></mp:jsModule>
        <mp:jsModule module="ojs/ojvalidation"></mp:jsModule>
        <mp:jsModule module="ojs/ojslider"></mp:jsModule>
        <mp:jsModule module="ojs/ojpagingcontrol"></mp:jsModule>
    </mp:jsLibrary>
  </mp:jsLibraries>

...

</mp:Integration>
```

3. Use the EDK to create your OPAR.

4. Import and Deploy your plug-in to Enterprise Manager.

| Element | Description |
|---|---|
| `mp:Integration` | The `mpcuiLibVersion` specified which version of the MPCUI library (and JET library) your UI will run against.<br><br>`<mp:Integration`<br>`mpcuiLibVersion="13.2.0.0.0"`<br><br>`xmlns:mp="http://www.oracle.com/`<br>`EnterpriseGridControl/`<br>`MpCuiIntegration">`<br><br>For your discovery library, set `integrationType` and `discoveryModule`.<br><br>`<mp:Integration`<br>`mpcuiLibVersion="13.2.0.0.0"`<br><br>`integrationType="discovery"`<br>`discoveryModule="DemoHostSample"`<br><br>`xmlns:mp="http://www.oracle.com/`<br>`EnterpriseGridControl/`<br>`MpCuiIntegration"`<br>`        >` |

| Element | Description |
| --- | --- |
| sourceContext | The Integration Metadata was designed so that you can reuse the Integration MXML file from a Flex implementation. The sourceContext bootstraps some of the old notation and limits the ammount of duplication necessary in the paths of the files contained in the plug-in. |

```
<mp:sourceContext>
  <mp:jsRoot path="js/libs/dhs"/>
  <mp:cssRoot path="css/dhs"/>
  <mp:bundleRoot path="rsc"/>
</mp:sourceContext>
```

| Element | Description |
| --- | --- |
| cssFiles | Each css file your application uses is specified here. |

```
<mp:cssFiles>
  <mp:cssFile
id="myCss" path="dhs.css"
version="13.1.0.1.0"/>
</mp:cssFiles>
```

| Element | Description |
|---|---|
| jsLibraries | This tag defines each JS library used, along with the dependencies each will require in the main.js file. |

```
<mp:jsLibraries>
  <mp:jsLibrary
id="pluginLib" path="libs/dhs/
demo_hostsample-min.js"

debugPath="libs/dhs/
demo_hostsample-debug.js"
version="13.2.0.0.0"
                  isDefault="true">
    <mp:jsModule module="ojs/
ojmodel"></mp:jsModule>
    <mp:jsModule module="ojs/
ojknockout"></mp:jsModule>
    <mp:jsPath ...></mp:jsPath>
    <mp:jsShim ...></mp:jsShim>
<!--
    There can
be only 1 default library.  Any
classes that aren't attached to
    another library will be
attached to the default library.

    If not the
default, you can associate
an activity with the library with:
      mp:jsPath id="activityId"

    and the
appropriate path will be added
for that activity or the explicit
    path can be set:
      mp:jsPath path="dhs/
MyController"

    Modules to
be added to the initial require
clause (where the app is launched)
    can be added here:
      mp:jsModule
module="ojs/ojtable"

    Shims may also be specified:
      mp:jsShim name="myshim"
exports="myshim" deps="jquery,
ojs/ojcore"

-->
```

| Element | Description |
|---|---|
| resourceBundles | |
| | ```
<mp:resourceBundles>
  <mp:MpBundle name="demoUiMsg"
path="oracle.samples.xohs.rsc"
isDefault="true"/>
  <mp:MpBundle name="demoJobMsg"
path="oracle.samples.xohs.rsc"/>

</mp:resourceBundles>
``` |
| activities | All pages, dialogs, trains, etc used by your UI. This section can be taken directly from the MXML Integration file and copied into the Integration Metadata. |

# About Logging

The following sections discuss the logging options for MPCUI.

## Add Logging to your Code

Use the logging facility (MpLog) to log messages from your code.

While logging can be useful in situations where diagnostics are necessary, it has a cost in terms of code size and overhead. Therefore, use logging with care.

Perform logging by calling one of several MpLog methods (such as debug, info, error, or fatal). The methods accept a message string and an optional list of parameters that must be substituted.

To substitute parameters, indicate the parameter location using {#} format:

```
  MpLog.debug("The metric {1} was not found for the target {2}.", metricName,
targetName);
```

The message generated for this log statement appears in the following log output:

```
2017-04-22 11:10:17 [MpCui] DEBUG The metric CPU was not found for the target
MYHOST
```

The level (info, debug, error, fatal) allows the user to enable log output for different classes of messages.

- By default, all error and fatal messages are sent to the log output location.

- The info and debug level messages are only sent if these levels are explicitly enabled.

Furthermore, you can direct the messages for each level to different output locations. There are three possible log locations:

- EMLOG: messages are sent to the Enterprise Manager application logs

- CONSOLE: messages are sent to the console log, accessible through the browser developer tools

# Options for Capturing Log Output

The options for capturing log output depend on your implementation:

- Running MPCUI from NetBeans
- Running MPCUI from the Enterprise Manager Console

## Running MPCUI from NetBeans

When you are developing MPCUI using NetBeans, the log messages appear in the console window at the bottom of the NetBeans integrated development environment (IDE) by default.

To change these logging settings:

1. Open the data/mpCuiProperties.xml file..

2. Locate the `loglevel` element:

```
<!-- Logging
     level: DEBUG, INFO, ERROR, FATAL, WARN (or ALL)
     output location: CONSOLE, EMLOG
     format: level,output;level,output (e.g. DEBUG,CONSOLE;ERROR,EMLOG)
-->
<loglevel>ALL,EMLOG</loglevel>
```

3. Modify the `loglevel` element as required.

## Running MPCUI from the Enterprise Manager Console

After the plug-in is deployed, the settings for logging are detected from the HTTP request. The default setting is FATAL,CONSOLE;ERROR, CONSOLE.

The end user can modify the settings as follows:

1. Append the following to the URL in the address bar of the browser:

   `&loglevel=ALL,EMLOG`

2. You can substitute ALL,EMLOG with any valid logging settings, such as ERROR,CONSOLE and so on.

3. For diagnostic situations, add the following to the end of the URL:

   `&loglevel=ALL,CONSOLE`

**Figure 9-26    Viewing Log Messages**



# Development Environment Options

When building a custom UI for your plug-in, you have the following development environment options:

- NetBeans

  NetBeans is a free IDE which you can download from **http://netbeans.org**. For more information about using NetBeans, see Overview of MPCUI Metadata Elements.

- An IDE of your choosing

  There is no limitation on which IDE you use to develop your UI.

## Developing MPCUI in NetBeans

This section describes the process to follow when building a UI using the MPCUI libraries and NetBeans. These steps assume the use of the sample application provided with the EDK referred to as the Demo Host Sample (or demo_hostsample). As with many development activities it is often easiest to start with a working example to understand how the provided APIs work and how to use them to accomplish higher-level use cases.

To simplify the process for developing your custom UI, build and run the custom UI project from NetBeans without having to redeploy the plug-in to Enterprise Manager after each change. When running from NetBeans, your UI will not have access to the other Enterprise Manager features and pages available to the console, but you will be able to exercise your UI to ensure that it is functioning correctly before deploying it as part of your plug-in.

## Setting up the Demo Sample Project

To set up the demo sample project:

1.  Locate the org-oracle-demo_hostsample.nbm file in the EDK distribution.

2.  Copy this file to a location on your Windows system where you installed NetBeans.

3.  From NetBeans, from the **Tools** menu, select **Plugins**.

4.  Select the **Downloaded** tab and click the **Add Plugins…** button. Navigate to the .nbm file and click **Open**.

5.  In the Downloaded tab, select the Oracle MPCUI Demo Hostsample plugin and click **Install.**

6.  Click through the dialog that pops up and install the plug-in.

7.  Click the **New Project…** icon or the **File** menu and then **New Project…**

8.  Open the Samples folder and select HTML5/JavaScript. Select Oracle MPCUI Demo Hostsample Project. Click **Next** and then **Finish** on the next screen.

9.  This creates a Demo Hostsample project for you. Navigate to the js/dhs directory to see the JaS Controller files and the view/dhs directory to see the HTML pages that comprise the plug-in.

10. This process can be repeated for the Demo Hostsystem module, and the MPCUI Starter module, which is a project which has all of the required dependencies (JET, MPCUI, etc) but no page or controller implementations. The MPCUI Starter project will appear under HTML5/JavaScript rather than Samples>HTML5/JavaScript.

    For information about building the project's JS library, see Packaging the MPCUI Implementation With the Plug-in and Defining the MPCUI Application.

## Running Demo Sample MPCUI from NetBeans

> **Note:**
>
> One of the advantages of MPCUI is that it allows you to test your UI as part of the deployed plug-in or by running it from within NetBeans directly. This latter option makes iterative development much simpler, however it requires that at least one version of the plug-in is deployed to Enterprise Manager and that a target instance has already been discovered before attempting to run the UI.

After the Demo Sample plug-in has been deployed and you have created an instance of the Oracle MPCUI Demo Hostsample Project in NetBeans, then you can run the project from NetBeans.

1.  From the Navigator, select **demo_hostsample**.

2.  From the **Run** menu, select **Run Project**. To debug, you would right click on index-debug.html and select **Run File**.

    A browser window appears with a Management Server Connection login dialog.

> **✐ Note:**
>
> If the Management Server Connection dialog does not appear or if any other error appears, then verify that the project was imported correctly and verify that no errors appear in the Output>Browser Log tab that appears in the bottom of the NetBeans window.
>
> During normal operation, when the user accesses your UI through the Enterprise Manager console by going to a target home page, this dialog does not appear because the UI is running as an integral part of the Enterprise Manager console and is embedded in a console session.
>
> However, when running from NetBeans, your UI requires information to connect to the Enterprise Manager site where your plug-in has been deployed and where the target instance that you will manage is located. Enter the same information for host, port and credentials that you would use to connect to your running Enterprise Manager console. Use either http or https depending on your configuration; however you must ensure that the ports you supply are correct for the protocol supplied.

3. Below Target to Monitor, enter the target name and type (the internal type and not the displayed label) of a target instance associated with this plug-in. It must be a target that exists in Enterprise Manager already. If you are using the Demo Sample, then the target type is demo_hostsample, and the name is the target name you provided when creating the target instance.

4. Click **OK**.

   The Demo Sample home page appears and is populated with data.

> **✐ Note:**
>
> In this mode, the Enterprise Manager page decorations and the target context area do not appear at the top of the page but they will appear when you access the target home page from the Enterprise Manager console. A menu appears that allows you to exercise the multiple pages included in your custom UI.

## Elements of the Demo Sample UI

The following is a brief list of the components that make up the Demo Sample UI. For more comments that describe the purpose of each file and the items demonstrated in each file, see the source code.

```
demo_hostsample
    css/dhs
        dhs.css                     Style sheet for plug-in UI
    data/metadata
        demo_hostsample_uimd.xml    An abbreviated metadata file used to create
menu
                                    items in standalone mode
    data/metadata/stage
        demo_hostsample_uimd.xml    Full metadata file used to package and
deploy the
```

```
                                           plug-in
     em/mpcuiswf/loader/images        Any images included with the custom UI
go in this
                                       directory both in the running project and
in the
                                       stage/oms/metadata/mpcui directory.
     js/dhs
          HomePageController.js       All of the controllers for the HTML pages
in the
                                       plug-in go in a plug-in specific directory
under
                                       the js directory.
          ProcessesPageController.js
     view/dhs
          HomePage.html               The target homepage, contains the layout of
                                       the UI for the homepage
          ProcessesPage.html
     tools                            Contains the files and logic for building
the
                                       plug-in MPCUI Application into a single JS
                                       library containing all JS and HTML files.
```

## Updating the Demo Sample

As you modify and rebuild your UI in NetBeans, you can run or debug the UI directly from NetBeans as you make changes.

> **✎ Note:**
>
> If you use the Chrome browser, you should install the NetBeans Connector and the Knockoutjs context debugger in the Extensions. This will allow you to debug through the NetBeans IDE**.**

## Modifying the Deployed Plug-in

After you make changes to your UI in NetBeans, you can apply the changes to your plug-in so that you can also view the updates from a target home page within the Enterprise Manager console.

To do this, you must either create and deploy a new version of your plug-in or use the metadata registration service (MRS).

MRS allows you to apply incremental updates to your plug-in without creating and deploying a entirely new version. For more information about MRS, see Updating Deployed Metadata Files Using the Metadata Registration Service (MRS).

After you have modified your custom UI:

1. Rebuild the JS library.

2. FTP or copy the library (and MPCUI XML file containing the Integration metadata) to the server where your Enterprise Manager site is installed and where you deployed the Demo Sample plug-in originally.

3. Copy this file to the location where you created the plug-in staging directory:

   ```
   stage/oms/metadata/mpcui
   ```

> **Note:**
>
> There is an existing version of this file in the directory (or a subdirectory) along with an MPCUI metadata XML file.

4. Update the plug-in using the following command:

```
emctl register oms metadata -sysman_pwd sysman -pluginId oracle.sysman.xohs
-service mpcui -file demo_hostsample_uimd.xml
```

For information about the `emctl` command, see Updating Deployed Metadata Files Using the Metadata Registration Service (MRS).

## Setting Up a NetBeans Project for MPCUI

To set up a NetBeans, you can create an empty project (the MPCUI Starter Project, which can be installed from the nbm file included with the EDK) or create a Demo Hostsample project to use as a template.

## Creating a NetBeans Project

If you are using the demo_hostsample project as a template (and some of these steps will apply for the MPCUI Starter Project), you must complete the following steps before you set up the NetBeans project for MPCUI.

1. Delete the contents of the following directories:

   • /opar

   • /rsc

   These directories provide support for deploying the sample plug-in, but are not appropriate to your new project.

2. In the data/metadata and data_metadata/stage directories, rename the demo_hostsample_uimd.xml file to targettype_mpcui.xml, where targettype is the name of your target type.

   Delete all the other contents of the data/metadata directory except targettype_mpcui.xml. The file in the data/metadata directory is a slimmed down version of the Integration Metadata used to create the menu items and run the UI in standalone mode. The XML file in data/metadata/stage is the fully populated file you would use to package with the plug-in.

3. From the data directory, edit the mpCuiProperties.xml file as follows:

   • Replace the OMS connection with the information for connecting to your Enterprise Manager server.

   ```
   <!-- Default OMS Connection -->
       <hostname>myhost.us.example.com</hostname>
       <port>7777</port>
       <emUser>sysman</emUser>
       <password>sysman_pasword</password>
   ```

   • Replace the `<metadata>` tag with the file name as created in step 2 (/metadata/*targettype*_mpcui.xml)

```
<!--             the filename that includes the mpcui meta-data that will
be included
in the plug-in. This is used to populate the menus for testing of the
UI in standalone (FlashBuilder) mode
If not specified then a default filename of <targetType>_menu.xml will
be used. -->
        <metadata>../metadata/targettype_mpcui.xml</metadata>
```

4. You are ready to start developing your UI. HTML pages go in a plug-in specific directory in the view directory (for demo_hostsample, it's view/dhs). JS controller files go in a plug-in specific directory in the js directory (for demo_hostsample, it's js/dhs).

# Home Page Customizations

Earlier versions of the Enterprise Manager extensibility framework supported the ability to customize the default Enterprise Manager home page by:

• setting a set of charts to display on the home page

• defining a series of related links to display on the home page

For Enterprise Manager release 13.1, this is no longer supported. A new custom UI implementation based on HTML/JS and JET will have to be created to replace this previous implementation.

# Accessibility Guidelines

The MPCUI framework is designed to support a user interface that complies with the Oracle Global HTML Accessibility Guidelines (OGHAG). This section provides information about accessibility standards for your UI implementation.

Also, JET provides guidelines and information to help with the implementation of JET UIs (on which MPCUI is based) to meet accessibility standards. For more information, see the JET Accessibility Page: https://www.oracle.com/webfolder/technetwork/jet/index.html.

## Accessibility Options in Enterprise Manager

Enterprise Manager provides the end user with the ability to set options for accessibility including a screenreader option. The MPCUI framework is aware of these settings and makes them available to you in your JS code (see the `oracle.sysman.emx.util.AdaSettings` in the API reference).

Typically you do not have to check for these settings because MPCUI automatically renders accessible components when the end user sets their account to require an accessible user interface. Among other things, this replaces charts with an accessible view of the same data.

## Summary of Critical Issues

When constructing an accessible MPCUI custom UI, consider the following items:

• Use MPCUI Pages, dialog and components

These components include accessibility support.

- Set Name and Description

  For components that require additional text description (such as images).

- Avoid conveying information using color.

# Localization Support

To provide support for localized text resources, you must use strings in the custom UI. To do this, you must:

- Register Bundles
- Reference Strings from HTML (Page, Dialog Definitions)
- Access Strings from JavaScript (Controller Code)

## Register Bundles

To use resource properties files in MPCUI, you must register resource bundles in the integration metadata. Include a block such as the following:

```
<mp:resourceBundles>
<mp:MpBundle name="demoUiMsg" path="oracle.samples.xohs.rsc" isDefault="true" />
<mp:MpBundle name="demoJobMsg" path="oracle.samples.xohs.rsc" />
</mp:resourceBundles>
```

As this shows, you can break up your resources into more than one bundle, and mark one bundle as the default bundle. This simplifies access to strings in this bundle throughout the rest of the UI code. The path attribute must be consistent with the path where the properties files were added to the Properties JAR file. For information about the Properties JAR file, see Package Resource Bundles.

## Reference Strings from HTML (Page, Dialog Definitions)

To reference a string in a page or dialog class in HTML, the getString and getBundleString methods are provided. The getString method retrieves strings from the default resource bundle, and the getBundleString method retrieves strings from any bundle registered in the Integration metadata.

The `getString` method is used as follows:

```
<mp-section id="configurationRegion" params="title: getString('CONFIGURATION')"
          style="height:60%;width:100%" ></mp-section>
```

This method locates a string with the key "CONFIGURATION" in the default resource bundle (demoUiMsg) and uses it as the title of this inner region. If the string cannot be found, then the key (CONFIGURATION) is shown.

The getBundleString method is used as follows:

```
<mp-info-item id="currentLoad" params="label: getBundleString('demoJobMsg',
'JOBLOAD'),value: respData.result.getString('','Load')"> </mp-info-item>
```

The first parameter to `getBundleString` specifies the bundle from which to retrieve the string.

## Access Strings from JavaScript (Controller Code)

To access strings from the JavaScript code, use either the Util.getString method or the Util.getBundleString method:

```
var str = RscUtil.getString("CONFIGURATION"); // retrieves string from default
bundle
var str2 = RscUtil.getBundleString("demoJobMsg", "JOBLOAD"); // retrieves string
from named bundle
```

# Providing Online Help

If you want to include online help for your customized UI pages, package the help JAR files in the following directory:

*plugin_stage*/oms/online_help

For an example of a help JAR file, see the plugin_sample_help.jar in the /oms/online help directory of the demo_hostsample example in the EDK.

# Migrating From Flex to HTML/JS/JET

This manual is a catalog of useful conversions between components and code patterns in Flex to the components and code patterns implemented using JavaScript (JS) and HTML, utilizing Oracle's JavaScript Extension Toolkit (JET). Many of the custom Flex components and ActionScript (AS) classes have corresponding components and classes built on the JS side. In many cases it will be a direct conversion from one to the other. In addition, the framework supported classes were nearly all reimplemented into JS, so all of the same structures and utilities will continue to exist and operate in the same way as before.

In the current Flex-based framework, you have MXML files backed by ActionScript. These are compiled into a SWF file and packaged with your plug-in to provide a custom UI for your target.

In the new JET-based framework, you will have HTML files backed by JavaScript. These are combined and minimized into a JS library and packaged with your plug-in to provide a custom UI for your target.

- MXML files will be converted into HTML files
- ActionScript files will be converted into JS files

An MXML file representing a page in your plug-in will be converted into an HTML page. The same MVC framework that existed in the Flex-based framework has been brought forward into the JET-base framework, so the controller AS file backing your MXML page will be converted into a controller JS file. All of the same web services with all of the same APIs will be supported.

The Demo Hostsample example has been directly cut over from Flex to this new, JET-based implementation.

Included in the MPCUI Framework is a set of custom tag (through KnockoutJS) implementations which mirror tags available in Flex. A programmatic migration path is not possible between Flex and HTML/JS, so the custom tag set cuts down on the

learning curve and the effort required to convert a Flex implementation to an HTML/JS one.

## Application Structure

In Flex, there is the ability to create a Page class and a Page Controller class from which to extend to provide a Model View Controller (MVC) structure to the application. In JavaScript and HTML, the implementation will be different, but the MPCUI has implemented a very similar structure which repeats this same MVC in an HTML/JS application.

## Model

The model is accessible through the controller to set values, and is accessible through the View to display values. In a typical HTML page, this data flow is not dynamic. However, through the use of the KnockoutJS library, any value set on the model:

```
var configData = { cpuModel:  info.getString(0, "CPU Model"),
                       osVersion: info.getString(0, "OS Version") };
  this.page.setModel("configData", configData);
```

becomes dynamic. Any integrator using JET or the MPCUI is encouraged to research the libraries utilized to create their UI, but in this case, you will not have to deal directly with Knockout observables if you don't choose to. Any object set to the page model is made dynamic automatically, so that you can access it in the HTML page:

```
<mp-info-item id="osVersion" params="label : getString('OS_VERSION'),
                                 value : model().configData().osVersion
"></mp-info-item>
```

without any additional work.

## Page (View)

The Page is now an HTML page. The custom tags implemented for MPCUI give the page its structure and will cut down on the overall content in the HTML file created for a plug-in.

```
<mp-data-services>
    <mp-sql-data-service id="ids" params="queryID:'INSTANCE_INFO',
properties:props('TARGET_GUID',appModel.target.guid)">
    </mp-sql-data-service>
    <mp-metric-values-data-service id="procData"
params="flattenData:true,
               targetName:appModel.target.name,
targetType:appModel.target.type,
               metricName:'CPUProcessorPerf', columns:['CPUIdle'],
               timePeriod:'CURRENT', interval:15">
    </mp-metric-values-data-service>
    <mp-metric-values-data-service id="processorData"
params="flattenData:true,
               targetName:appModel.target.name,
targetType:appModel.target.type,
```

```
                    metricName:'CPUProcessorPerf',
                    columns:['CPUIdle','CPUUser','CPUSystem','CPUIOWait'],
                    timePeriod:'CURRENT', predicate:model().processorFilter
">
    </mp-metric-values-data-service>
    <mp-avail-data-service id="ads"
params="targetName:appModel.target.name,
                    targetType:appModel.target.type, days:1">
    </mp-avail-data-service>
    <mp-association-data-service id="asc"
params="targetName:appModel.target.name,
                    targetType:appModel.target.type, assocTypes:
['hosted_by']">
    </mp-association-data-service>
    <mp-metric-values-data-service id="respData"
params="flattenData:true,
                    targetName:appModel.target.name,
targetType:appModel.target.type,
                    metricName:'Response', columns:['Load'],
                    timePeriod:'LAST_HOUR', flags:
'COUNTERS_FOR_RATE'">
    </mp-metric-values-data-service>
</mp-data-services>

    <!--
        The example below also demonstrates the ways data may be bound
to a UI component included in the page:

        1. Data Service Reference
        2. Global/Application Model Reference
        3. Page Model Reference
        4. Set Directly from Controller

        InfoDisplay/InfoItem click handling

        1. if a dataProvider specified for the InfoDisplay, then a
global click handler can be set for the entire component
        2. a click handler can be set for each InfoItem
        3. a click handler can call the invokeActivity method passing
an activity id and a bean (input context)
        4. a destination can be set for each InfoItem, and set to a
String that is an activity id
        5. a destination can be set for each InfoItem, and the
destination can be an actual Activity object constructed in the
controller
    -->
<mp-page>
    <mp-row>
        ...
    </mp-row>
</mp-page>
```

You can see that the data service tags are still available as they were in Flex, and the HTML content starts with the <mp-page> tag.

# Page Controller

This is the start of your topic.The Page Controller is implemented in JavaScript as a "class." There is no notion of class in JS, and no strong inheritance model if you do create JS objects or classes. We use the prototypal inheritance available in JS, along with require to structure a single JS file as a module and its own "class."

When creating a new Page Controller, your new class will extend from the ActivityController class provided by the MPCUI framework. At the top of any class, there is a define([])function() {} block. This is the require library notation that makes this one class/one file structure possible. It also serves as a de facto import block. Any of the classes used in the current class would appear in the define-function block at the top of the file.

```
define([
    "emx/intg/ActivityController",
    "emx/intg/InputParam",
    "emx/intg/UrlEm",
    "emx/MpLog",
    "emx/util/TargetContext",
    "emx/model/TargetFactory",
    "emx/service/util/AssociationDataService",
    "emx/service/metricDataService/MetricCollectionTimePeriod",
    "emx/util/Constants",
    "emx/util/Util",
    "emx/service/sqlQuery/BulkSqlQuery",
    "emx/util/RscUtil",
    "emx/intg/ActivityDef"
    ],
  function(
    ActivityController,
    InputParam,
    UrlEm,
    MpLog,
    TargetContext,
    TargetFactory,
    AssociationDataService,
    MetricCollectionTimePeriod,
    Constants,
    Util,
    BulkSqlQuery,
    RscUtil,
    ActivityDef
  ) {

    /**
    * Each page in the plugin UI will typically have a controller class
that extends
    * ActivityController. The controller contains the functions that
populate data to
    * be shown in the page and respond to events in the page (button or
link clicks for
    * example).  The controller should include an init method that will
be called when
```

```
     * the page is being initialized.  This method can be used to setup
any data that
     * will be displayed in the page.
     */
    function HomePageController() {
        this.Init();
    };
    oj.Object.createSubclass(HomePageController, ActivityController,
"HomePageController");

    HomePageController.prototype.Init = function() {
        HomePageController.superclass.Init.call(this);
    };

    HomePageController.prototype.page;

    HomePageController.prototype.destroy = function(page) {
        // do any cleanup of view/model here
    };
    HomePageController.prototype.initComplete = function(page) {
        // do any setup which requires the page and its components to
be fully loaded
    };

    /*
     * initialize the model for the page; data services declared as
part of the page
     * will automatically be loaded and initialized by the framework
and don't need
     * to be initialized here in any way
     *
     * NOTE: the refresh parameter is added in 13.1 allowing the
controller to know
     * if this is the first pass through the page or an incremental
refresh triggered
     * by the page-level refresh button
     */
    HomePageController.prototype.init = function(page, refresh) {
        this.page = page;
        ...
    };
    ...
    return HomePageController;
});
```

The class has a constructor. The parent class is specified in
the oj.Object.createSubclass call and the object which is created by this class
definition is returned at the bottom of the file. The APIs which could be extended
in the Flex version of this class can also be extended in the JS version.

Any function put on the class prototype: HomePageController.prototype.init is available to any instance of the class and can be referenced on the created object. Any function put directly on the class is static and is referenced with the class name.

```
HomePageController.prototype.myFunc = function() {
    };

    HomePageController.myStaticFunc = function() {
    };

    var hpc = new HomePageController();
    var fVal = hpc.myFunc();
    var staticFVal = HomePageController.myStaticFunc();
```

## Converting ActionScript to JavaScript

To mimic a class structure in JavaScript, MPCUI uses **RequireJs** to utilize the Asynchronous Module Definition (AMD) API in conjunction with JS prototypal inheritance. The result is the ability to break JS up into the same file structure as it was in AS.

| ActionScript | JavaScript | Notes |
|---|---|---|
| package ... import list | `define([ "emx/intg/ ActivityController", . .. ], function( ActivityController, .. . ) { // Class Definition });` | NA |
| public class X extends Y | `oj.Object.createSubclas s(X, Y, "X");` | NA |
| public var p:Page; | `X.prototype.p;` | Specifying a member of a class requires you to put it on the prototype in JS. |
| super.init(p) | `X.superclass.init.call( this, p);` | NA |
| p = pg as X; (casting pg to be class X) | `p = pg;` | Not necessary to cast in JS |
| NA | `page.initModel(["cpuMod el", "osVersion"]);` or `page.setModel("cpuModel ", "");` `page.setModel("osVersio n", "");` | Not necessary in Flex, but in JS, all model properties have to be initialized even if they don't have a value to begin with |

| ActionScript | JavaScript | Notes |
|---|---|---|
| ApplicationContext.getTargetContext(); | `TargetContext.getTargetContext();` | NA |
| Callbacks in Flex are as easy as specifying the function name only | In JS, you would use one of:<br>• `page.cb(functionName)`<br>• `Util.createProxy(this, this.functionName)`<br>• `oj.Object.createCallback(this, this.functionName)` | In JavaScript, the context in which you run a function is never assured. You will want to create a closure to ensure that the "this" in the function being called is the value the function expects. |
| page.model["loadDataSource"] | `page.getModel("loadDataSource");` | NA |

## Converting Flex Tags to MPCUI Custom HTML Tags

With the new JET-based framework, the custom components are supported as both **knockout** **custom bindings** (<div data-bind="..."></div>) and as knockout **custom elements** (<mp-info-display>, custom elements mean custom tags as well). Using the custom bindings, you would be responsible for managing all aspects of the layout of the page. For the custom elements, the layout is built into the elements and there are elements specifically crafted to mimic the behavior of the Flex layout options.

## Data Services

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| <mp:services> | <div class="dataServices"> | <mp-data-services> |

In the HTML page, this tag and all of the data services under it will come before the page tag in the page.

## SQL Data Service

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| ```<mp:SQLDataService id="ids" queryID="INSTANCE_INFO " properties="{props('TA RGET_GUID', appModel.target.guid)} "/>``` | ```<div id="ids" data- bind="mpSqlDataService : { queryID:'INSTANCE_INFO ', properties:props('TARG ET_GUID', appModel.target.guid) }"> </div>``` | ```<mp-sql-data-service id="ids" params="queryID:'INSTA NCE_INFO', properties:props('TARG ET_GUID', appModel.target.guid)" > </mp-sql-data-service>``` |

> **Note:**
>
> There is a closing "</div>" and a closing "</mp-sql-data-service>" tag in the HTML examples. It is strongly recommended that you adopt this technique for your HTML work. In the case of custom elements, this is a requirement. No custom elements can be self-closing (".../>").

## Metric Values Data Service

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:MetricValuesDataSe rvice id="procData" flattenData="true"`<br><br>`targetName="{appModel. target.name}"`<br><br>`targetType="{appModel. target.type}"`<br><br>`metricName="CPUProcess orPerf"`<br><br>`columns="{['CPUIdle']} "`<br>`timePeriod="REALTIME" interval="15" />` | `<div id="procData" data- bind="mpMetricValuesDa taService : {`<br>`    flattenData:true,`<br><br>`targetName:appModel.ta rget.name,`<br><br>`targetType:appModel.ta rget.type,`<br><br>`metricName:'CPUProcess orPerf',`<br>`    columns: ['CPUIdle'],`<br><br>`timePeriod:'CURRENT', interval:15 }">`<br>`</div>` | `<mp-metric-values- data-service id="procData"`<br><br>`params="flattenData:tr ue,`<br><br>`targetName:appModel.ta rget.name,`<br><br>`targetType:appModel.ta rget.type,`<br><br>`metricName:'CPUProcess orPerf',`<br>`    columns: ['CPUIdle'],`<br><br>`timePeriod:'CURRENT', interval:15">`<br>`</mp-metric-values- data-service>` |

## Association Data Service

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:AssociationDataSer vice id="asc"`<br><br>`targetName="{appModel. target.name}"`<br><br>`targetType="{appModel. target.type}"`<br><br>`assocTypes="{['hosted_ by']}" />` | `<div id="asc" data- bind="mpAssociationDat aService : {`<br><br>`targetName:appModel.ta rget.name,`<br><br>`targetType:appModel.ta rget.type,`<br>`    assocTypes: ['hosted_by'] }">`<br>`</div>` | `<mp-association-data- service id="asc"`<br><br>`params="targetName:app Model.target.name,`<br><br>`targetType:appModel.ta rget.type,`<br>`    assocTypes: ['hosted_by']">`<br>`</mp-association-data- service>` |

## Availability Data Service

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:AvailDataService id="ads"`<br><br>`targetName="{appModel.target.name}"`<br><br>`targetType="{appModel.target.type}" />` | `<div id="ads"`<br>`    data-bind="mpAvailDataService : {`<br><br>`targetName:appModel.target.name,`<br><br>`targetType:appModel.target.type,`<br>`    days:1 }">`<br>`</div>` | `<mp-avail-data-service id="ads"`<br><br>`params="targetName:appModel.target.name,`<br><br>`targetType:appModel.target.type,`<br>`    days:1">`<br>`</mp-avail-data-service>` |

## Page

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| mp:Page | `<div class="mp-page-content mp-flex">` | `<mp-page>` |

This tag will represent the topmost content container in the HTML file. Data services, if there are any in the page, would be before the page tag, but the page tag is the start of the content of the page. This custom binding example has 2 classes on it. The "mp-page-content" sets the height and width to 100% so the page will be set to fill all of the available space. This is important in HTML for 2 reasons:

1. HTML components will try to fill as little space as possible so if you don't tell it 100%, it won't expand out to fill the space and

2. for the layout engine to respect the correct height and width, they have to be set on every component in the DOM.

If the parent isn't set, HTML can get confused by a percentage, so it's safest to set it all the way down the page.

The other class specified relates to the layout engine supported by JET. The Flexible Box Layout (or flexbox) is documented here. The "mp-flex" class initializes a flexbox container for the whole page. The flexbox layout engine is used for the whole page. This, in addition to an easier to read page, is one of the benefits of using the custom element. Flexbox support is built into each custom element.

## tabOrder

Not supported in the HTML tag.

## Model Reference

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| {model.relatedHostType} | model().relatedHostType | model().relatedHostType |

The model is setup automatically for you on the page as an object, so you can reference it in the HTML similarly to how you did in your MXML page. As an observable, you reference the model with parentheses: model() and whatever id you used to save the data (this is also automatically made into an observable so any changes to model().id will automatically be consumed by the HTML page).

## HBox

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| mx:HBox | <div class="mp-flex mp-flex-row"> | <mp-row> |

If the height or width are not specified, they will automatically be set to 100% in the custom element case.

**Example**:

```
<mp-row style="height:33%">
```

will result in a row taking up a third of the parent's space. width will automatically be set to 100%

## VBox

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| mx:VBox | <div class="mp-flex mp-flex-col"> | <mp-column> |

If the height or width are not specified, they will automatically be set to 100% in the custom element case.

**Example**:

```
<mp-column style="width:50%">
```

will result in a column taking up half of the parent's space. height will automatically be set to 100%

## Region

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:Region`<br>`id="summaryRegion"`<br><br>`title="{getString('SUM`<br>`MARY')}"`<br>`  height="50%"`<br>`width="100%" >` | `<div`<br>`style="width:100%;heig`<br>`ht:100%"`<br>`  data-bind="mpPanel:`<br>`{title : '',`<br>`    headerBorder :`<br>`false}" >`<br>`  <div`<br>`id="summaryRegion"`<br>`style="height:50%"`<br>`    data-`<br>`bind="mpSection: {`<br>`      title :`<br>`getString('SUMMARY'),`<br>`      level : 2,`<br>`headerBorder : true}">` | `<mp-panel`<br>`params="title: '',`<br>`headerBorder: false">`<br>`  <mp-section`<br>`id="summaryRegion"`<br><br>`style="height:50%"`<br>`    params="title :`<br>`getString('SUMMARY'),`<br><br>`      level : 2,`<br>`headerBorder : true">` |

## InnerRegion

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:InnerRegion`<br>`id="statusRegion"`<br><br>`title="{getString('STA`<br>`TUS')}"`<br>`  height="40%"`<br>`width="100%" >` | `<div`<br>`id="statusRegion"`<br>`  style="height:40%"`<br>`  data-`<br>`bind="mpSection : {`<br>`    title :`<br>`getString('STATUS')}"`<br>`>` | `<mp-section`<br>`id="statusRegion"`<br>`  style="height: 40%"`<br>`  params="title :`<br>`getString('STATUS')" >` |

## InfoDisplay

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:InfoDisplay width="100%" height="100%">` | `<table data-bind="mpInfoDisplay">` | `<mp-info-display>` |

## InfoItem

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:InfoItem id="currentStatus"`<br><br>`label="{getString('CUR RENT_STATUS')}"`<br><br>`value="{ads.currentSta tus}"`<br><br>`image="{ads.currentSta tusIcon}"`<br><br>`click="invokeActivity( Constants.PAGE_AVAILAB ILITY,`<br><br>`bean(Constants.P_TARGE T_NAME,`appModel.target.na me`,`<br><br>`Constants.P_TARGET_TYP E,`<br><br>`appModel.target.type)) ;" />` | `<tr id="currentStatus"`<br>`  data-bind="mpInfoItem : {`<br>`    label : getString('CURRENT_STA TUS'),`<br>`    value: ads.currentStatus,`<br>`    image : ads.currentStatusIcon,`<br>`    destination : '#'}">` | `<mp-info-item`<br>`  params="label : getString('CURRENT_STA TUS'),`<br>`    value: ads.currentStatus,`<br>`    image : ads.currentStatusIcon,`<br>`    destination : '#'">` |
| `<mp:InfoItem id="currentLoad"`<br><br>`label="{getString('CPU _LOAD')}"`<br><br>`value="{respData.resul t.getString('','Load') }"`<br><br>`imageRenderer="{appMod el.renderer('CHECK_MAR K',`<br><br>`bean('type','number',' warning',`<br><br>`'0.1','critical','0.4' ))}" />` | `<tr id="cpuLoad"`<br>`  data-bind="mpInfoItem : {`<br>`    label : getString('CPU_LOAD'),`<br>`    value : respData.result.getStr ing('','Load'),`<br>`    imageRenderer: rendererFactory.get('C HECK_MARK',`<br>`bean('type','number',' warning',`<br>`'0.1','critical','0.4' )) }">` | `<mp-info-item id="cpuLoad"`<br>`  params="label : getString('CPU_LOAD'),`<br>`    value : respData.result.getStr ing('','Load'),`<br>`    imageRenderer: rendererFactory.get('C HECK_MARK',`<br>`bean('type','number',' warning',`<br>`'0.1','critical','0.4' )) ">` |

## Link

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:Link id="allReports"`<br><br>`label="{getString('ALL _REPORTS')}"`<br><br>`destination="{model.al lReportsLink}" />` | `<a id="allReports"`<br>`  data-bind="mpLink: {label: getString('ALL_REPORTS '),`<br>`  destination: model().allReportsLink }" />` | `<mp-link id="allReports"`<br>`  params="label: getString('ALL_REPORTS '),`<br>`  destination: model().allReportsLink ">` |

## Dialog

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:Dialog`<br>`  xmlns:mx="http:// www.adobe.com/2006/ mxml"`<br>`  xmlns:mp="http:// www.oracle.com/mpcui"`<br>`  width="550" height="530"`<br>`  title="Credentials">` | `<div data- bind="mpDialog : {`<br>`  height:620,`<br>`width:560,`<br><br>`title:'Credentials' }" >` | `<mp-dialog params="mpDialog : {`<br>`  height:620,`<br>`width:560,`<br><br>`title:'Credentials' }" >` |

There are a couple of custom elements which follow a slightly different model for passing parameters. Instead of listing out each parameter, they are listed out as properties of a single parameter, such as mpDialog.

## Train

## TrainContainer

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| ```<mp:TrainContainer id="createTrainContainer" width="100%" height="100%" trainId="addNewFSCreateTrain" trainDone="{controller.trainDone(event)}" />``` | ```<div style="width:100%;height:100%" data-bind="mpTrainContainer: { trainId: 'addNewFSCreateTrain', trainDone: controller.trainDone}">``` | ```<mp-train-container params="trainId: 'addNewFSCreateTrain', trainDone: controller.trainDone">``` |

## TrainStepPage

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| ```<mp:TrainStepPage xmlns:mp="http://www.oracle.com/mpcui" xmlns:mx="http://www.adobe.com/2006/mxml" width="100%" height="100%"``` | ```<div style="width:100%;height:100%" >``` | ```<mp-train-step-page>``` |

As with the mp-page tag, this is mostly a convenience for consistency.

## Table

The MP table custom binding and element extend from the JET ojTable and will support all of the properties documented on the JET website. In addition, the following properties will be supported:

- targetName
- targetType
- metricName
- metricColumns
- keys
- timePeriod

- interval

- dataService

- customDataSource

- dataProvider

This support mirrors what is available in Flex for the acquisition of target instance data for display in a table.

## API Changes

| Property Name | New Property Specification | Notes |
|---|---|---|
| paging | paging: 'on' | The paging control in JET is a separate custom binding which attaches to the same data source as the table. |
| | | The table custom binding manages that data source behind the scenes, so this indicates to the custom binding that paging is included. |
| | | Additionally, in the paging binding, tableId is accepted as an input to bind that to a table. |
| dataUpdateListener | dataUpdateListener: cb(controller.updateTable) | |
| mpRenderer | columns: { [ { mpRenderer: rendererFactory.get(...) } ] } | All of the same MP renderers are supported in this release. |

## Examples

**Table (using renderers)**

| Flex Tag | Custom Binding |
|---|---|
| ```<table id="incidentsTable" style="height:100%;width:100%"   data-bind="mpTable: {     customDataSource: model().incidentsData,     columns: [       {displayIndex: 0, headerText: 'Summary', field: 'summary',         id: 'summary', sortable: 'enabled',         headerStyle: 'text-align: left; white-space:nowrap;width:200px',         style: 'text-weight:bold;'},       {displayIndex: 1, headerText: 'Target', field: 'target',         id: 'target',         mpRenderer: rendererFactory.get('TARGET_TYPE') ,         style: 'text-align: center',         headerStyle: 'text-align:center;white-space:nowrap;width:200px'},       {displayIndex: 2, headerText: 'Severity',         field: 'severity', id: 'severity',         mpRenderer: rendererFactory.get('SEVERITY'),         style: 'text-align: center'},       {displayIndex: 3, headerText: 'Status',         field: 'status', id: 'status'},       {displayIndex: 4, headerText: 'Escalation',         field: 'escalation', id: 'escalation'},       {displayIndex: 5, headerText: 'Type', field: 'type', id: 'type'},       {displayIndex: 6, headerText: 'Last Update',         field: 'lastUpdate', id:``` | ```<mp-table id="incidentsTable" style="height:100%;width:100%"   params="mpTable: {     customDataSource: model().incidentsData,     columns: [       {displayIndex: 0, headerText: 'Summary', field: 'summary',         id: 'summary', sortable: 'enabled',         headerStyle: 'text-align: left; white-space:nowrap;width:200px',         style: 'text-weight:bold;'},       {displayIndex: 1, headerText: 'Target', field: 'target',         id: 'target',         mpRenderer: rendererFactory.get('TARGET_TYPE') ,         style: 'text-align: center',         headerStyle: 'text-align:center;white-space:nowrap;width:200px'},       {displayIndex: 2, headerText: 'Severity',         field: 'severity', id: 'severity',         mpRenderer: rendererFactory.get('SEVERITY'),         style: 'text-align: center'},       {displayIndex: 3, headerText: 'Status',         field: 'status', id: 'status'},       {displayIndex: 4, headerText: 'Escalation',         field: 'escalation', id: 'escalation'},       {displayIndex: 5, headerText: 'Type', field: 'type', id: 'type'},       {displayIndex: 6, headerText: 'Last Update',         field: 'lastUpdate', id:``` |

| Flex Tag | Custom Binding |
| --- | --- |
| `'lastUpdate' } ]}" >`<br>`</table>` | `'lastUpdate' } ]}" >`<br>`</mp-table>` |

The renderers supported in the Flex version are still supported and referenced from HTML in much the same way they were in the Flex-based release.

**Table (with paging)**

| Flex Tag | Custom Binding |
| --- | --- |
| `<table id="processesTable"`<br>`style="height:80%;width:100%"`<br>`  data-bind="mpTable: {`<br>`    selectionMode: {row:`<br>`'single', column: 'multiple'},`<br>`    dataService:`<br>`'processesDataSource',`<br>`    paging: true,`<br>`    dataUpdateListener:`<br>`cb(controller.processesTableUpdate`<br>`d) }">`<br>`</table>`<br>`<div id="paging"`<br>`style="width:100%"`<br>`  data-bind="mpPagingControl: {`<br>`    tableId: 'processesTable',`<br>`    pageSize: 15}" class="oj-`<br>`table-panel-bottom">`<br>`</div>` | `<mp-table id="processesTable"`<br>`  params="`<br>`    mpTable: {`<br>`      selectionMode: {row:`<br>`'single', column: 'multiple'},`<br>`      dataService:`<br>`'processesDataSource',`<br>`      dataUpdateListener:`<br>`cb(controller.processesTableUpdate`<br>`d) },`<br>`    mpPagingControl: {`<br>`      pageSize: 15 }">`<br>`</mp-table>` |

Notice with the custom element, the **paging** property isn't in the mpTable object and the **tableId** is not in the mpPagingControl. These are detected and set automatically as a part of the initialization of the custom element.

## Chart

This MpCui component extends from the JET chart component, so the properties specified below are supported in addition to all of the properties documented on the JET website. Currently the same chart types which were supported in Flex will be supported in the JET version (with the corresponding JET chart type in parentheses):

- LineChart (type:'line')

- AreaChart (type:'area')

- BarChart (type: 'bar', orientation: 'horizontal')

- ColumnChart (type: 'bar')

- PieChart (type: 'pie')

You are welcome to use the other chart types JET offers, but only the above charts will support the following properties:

- targetName
- targetType
- metricName
- metricColumns
- keys
- timePeriod
- interval
- dataService
- customDataSource
- dataProvider

If you do wish to use another chart type, you can still use the mp-chart custom element or the mpChart custom binding and it will serve as a pass through, but you will have to populate your data as shown in the JET website.

## API Changes

| Property Name | New Property Specification | Flex Chart Type | Notes |
|---|---|---|---|
| title | title: { text: 'Title' } | ALL | |
| subtitle | subtitle: { text: 'Subtitle' } | ALL | |
| footnote | footnote: { text: 'Footnote' } | ALL | |
| colors | styleDefaults: { colors: [] } | ALL | |
| showLegend | legend: { rendered: 'off' } | ALL | on (default) or off |
| legendLocation | legend: { position: 'bottom' } | ALL | Flex: left, right, top, bottom; JET: auto (default), start, end, top, bottom |
| legendDirection | | ALL | Not supported |
| selectionMode | selection: 'single' | ALL | Flex: single, multiple; JET: none, single, multiple |
| **mpSeries** | mpSeries: {...} | ALL Series | Supported in custom binding/element<br><br>This is an array of series objects which also supports a few Flex properties in addition to all of the documented JET series object properties. |

| Property Name | New Property Specification | Flex Chart Type | Notes |
|---|---|---|---|
| **displayName** | mpSeries: { [ { displayName: 'Name' } ] } | ALL Series | Supported in custom binding/element |
| **dataFunction** | mpSeries: { [ { dataFunction: cb(controller.myDatafunction } ] } | ALL Series | Supported in custom binding/element |
| selectable | | ALL Series | Not supported |
| **Line/Area Chart** | | | |
| axisRenderers | | Line, Area, Bar, Column | Accessible through the **xAxis** and **yAxis** property objects |
| xTitle | xAxis: { title: 'Title' } | Line, Area, Bar, Column | |
| yTitle | yAxis: { title: 'Title' } | Line, Area, Bar, Column | |
| yMax | yAxis: { max: 35 } | Line, Area | |
| yMin | yAxis: { min: 5 } | Line, Area | |
| startDate | | Line, Area | Not supported |
| endDate | | Line, Area | Not supported |
| **dataCapacity** | dataCapacity: 250 | Line, Area | Supported in custom binding/element (default depends on the time period selected) |
| **showCumulativeLine** | showCumulativeLine: 'true' | Line, Area | Supported in custom binding/element |
| **warningLine** | warningLine: { value: 5, showOnTop: false } | Line, Area, Bar, Column | Supported in custom binding/element |
| **alertLine** | alertLine: { value: 15, showOnTop: true } | Line, Area, Bar, Column | Supported in custom binding/element |
| fillFunction | | LineSeries, AreaSeries | Not supported |
| sortOnXField | | LineSeries, AreaSeries | Not supported |
| interpolateValues | | LineSeries, AreaSeries | Not supported |
| xField | | LineSeries, AreaSeries | Not supported |
| yField | | LineSeries, AreaSeries | Not supported |
| **Bar/Column Chart** | | | |
| **groupBy** | | Bar, Column | Supported in custom binding/element (byKey, byColumn) |
| seriesNames | | Bar, Column | Future |

| Property Name | New Property Specification | Flex Chart Type | Notes |
| --- | --- | --- | --- |
| sortBy | | Bar, Column | Future |
| seriesProperties | | Bar, Column | Not supported |
| labelPosition | styleDefaults: { dataLabelPosition: 'auto' } | Bar, Column | Flex: none, inside, outside; JET: auto (default), center, aboveMarker, belowMarker, beforeMarker, afterMarker |
| barWidthRatio | | Bar, Column | Not supported |
| maxBarWidth | styleDefaults: { maxBarWidth: 10 } | Bar, Column | |
| type | | Bar, Column | Flex: clustered, overlaid, stacked, 100%; JET: Not supported (stacking available) |
| offset | | BarSeries, ColumnSeries | Not supported |
| stacker | stack: 'on' | BarSeries, ColumnSeries | Allows only to turn it on or off |
| minField | | BarSeries, ColumnSeries | Not supported |
| labelField | | BarSeries, ColumnSeries | Not supported |
| labelFunction | | BarSeries, ColumnSeries | Not supported |
| **Pie Chart** | | | |
| labelPosition | styleDefaults: { sliceLabelPosition: 'auto' } | Pie | Flex: none, outside, callout, inside, insideWithCallout; JET: auto (default), none, outside, inside) |
| **selectedColumn** | | Pie | Supported in custom binding/element |
| **selectedKey** | | Pie | Supported in custom binding/element |
| innerRadius | styleDefaults: {pieInnerRadius: '0.5'}, | Pie | |
| textAlign | | Pie | Flex: left, right, center; JET: Not supported |
| explodeRadius | styleDefaults: {pieInnerRadius: '0.5', selectionEffect: 'explode'}, | PieSeries | JET: selectionEffect: explode, highlight, highlightAndExplode |
| perWedgeExplodeRadius | | PieSeries | Not supported |
| reserveExplodeRadius | | PieSeries | Not supported |

| Property Name | New Property Specification | Flex Chart Type | Notes |
|---|---|---|---|
| startAngle | | PieSeries | Not supported |
| maxLabelRadius | | PieSeries | Not supported |
| outerRadius | | PieSeries | Not supported |
| labelFunction | | PieSeries | Not supported |
| fillFunction | | PieSeries | Not supported |
| field | mpSeries: { [ { field: "myField" } ] } | PieSeries | Supported in custom binding/element |
| **nameField** | mpSeries: { [ { nameField: "myNameField" } ] } | PieSeries | Supported in custom binding/element |
| labelField | | PieSeries | Not supported |

This is an example of what using some of the above notation would look like, when put together:

| Flex Tag | Custom Binding |
|---|---|
| ```<br><div id="metDataCustBinding"<br>style="height:90%;width:100%"<br>  data-bind="mpChart: {<br>    type: 'line',<br>    title: {text: 'My Title'},<br>    subtitle: {text: 'My<br>subtitle'},<br>    dataSelection: 'multiple',<br>    emptyText: 'No data',<br>    yAxis: {title: 'Data', min:<br>0, max: 10},<br>    animationOnDisplay: 'auto',<br>    targetName: 'HostSample',<br>    targetType:<br>'demo_hostsample',<br>    metricName:<br>'CPUProcessorPerf',<br>    metricColumns:<br>['CPUUser','CPUSystem'],<br>    timePeriod: 'REALTIME',<br>    interval: 15}"><br></div><br>``` | ```<br><mp-chart id="metDataCustBinding"<br>style="height:90%;width:100%"<br>  params="mpChart: {<br>    type: 'line',<br>    title: {text: 'My Title'},<br>    subtitle: {text: 'My<br>subtitle'},<br>    dataSelection: 'multiple',<br>    emptyText: 'No data',<br>    yAxis: {title: 'Data', min:<br>0, max: 10},<br>    animationOnDisplay: 'auto',<br>    targetName: 'HostSample',<br>    targetType:<br>'demo_hostsample',<br>    metricName:<br>'CPUProcessorPerf',<br>    metricColumns:<br>['CPUUser','CPUSystem'],<br>    timePeriod: 'REALTIME',<br>    interval: 15}"><br></mp-chart><br>``` |

## Examples

### Line Chart

**Using customDataSource:**

| Flex Tag | Custom Binding |
|----------|----------------|
| ```<div id="lchart_from_custom" style="height:100%;width:100%"   data-bind="mpChart: {     type: 'line',     customDataSource: model().cpuChartData,     emptyText: 'No data',     animationOnDisplay: 'auto'}"> </div>``` | ```<mp-chart id="lchart_from_custom" style="height:100%;width:100%"    params="mpChart: {     type: 'line',     customDataSource: model().cpuChartData,     emptyText: 'No data',     animationOnDisplay: 'auto'}" > </mp-chart>``` |

from controller.init:

```
var cpuData = page.getModel("cpuChartData");
        if (cpuData == null) {
            cpuData = new CustomDataSource(["Sys/IO", "Idle %"], false,
true);
            page.setModel("cpuChartData", cpuData);
        }
```

**Using metric specification (with keys specified):**

| Flex Tag | Custom Binding |
|----------|----------------|
| ```<div id="lchart_from_metric2" style="height:100%;width:100%"   data-bind="mpChart: {     type: 'line',     dataSelection: 'multiple',     emptyText: 'No data',     animationOnDisplay: 'auto',     targetName: appModel.target.name,     targetType: appModel.target.type,     metricName: 'FilesystemPerf',     metricColumns: ['Utilization'],     keys: controller.page.keys([ [['MountPoint', '/']],      [['MountPoint', '/dev/shm']] ]),     timePeriod: 'REALTIME',     interval:60}"> </div>``` | ```<mp-chart id="lchart_from_metric2" style="height:100%;width:100%"   params="mpChart: {     type: 'line',     dataSelection: 'multiple',     emptyText: 'No data',     animationOnDisplay: 'auto',     targetName: appModel.target.name,     targetType: appModel.target.type,     metricName: 'FilesystemPerf',     metricColumns: ['Utilization'],     keys: controller.page.keys([ [['MountPoint', '/']],       [['MountPoint', '/dev/shm']] ]),     timePeriod: 'REALTIME',     interval:60}"> </mp-chart>``` |

**Showing reference lines:**

| Flex Tag | Custom Binding |
| --- | --- |
| <pre><div
id="lchart_from_metric_history"
style="height:100%;width:100%"
  data-bind="mpChart: {
    type: 'line',
    timeAxisType: 'enabled',
    dataSelection: 'multiple',
    emptyText: 'No data',
    title: {text: 'Free Swap
(KB)'},
    yAxis: {min: 0},
    warningLine : { value:
5000000, showOnTop: true },
    alertLine : { value: 1000000,
showOnTop: true },
    animationOnDisplay: 'auto',
    targetName:
appModel.target.name,
    targetType:
appModel.target.type,
    metricName: 'MemoryPerf',
    metricColumns: ['SwapFree'],
    timePeriod: 'LAST_DAY'}">
</div></pre> | <pre><mp-chart
id="lchart_from_metric_history"
style="height:100%;width:100%"
  params="mpChart: {
    type: 'line',
    timeAxisType: 'enabled',
    dataSelection: 'multiple',
    emptyText: 'No data',
    title: {text: 'Free Swap
(KB)'},
    yAxis: {min: 0},
    warningLine : { value:
5000000, showOnTop: true },
    alertLine : { value: 1000000,
showOnTop: true },
    animationOnDisplay: 'auto',
    targetName:
appModel.target.name,
    targetType:
appModel.target.type,
    metricName: 'MemoryPerf',
    metricColumns: ['SwapFree'],
    timePeriod: 'LAST_DAY'}">
</mp-chart></pre> |

## Area Chart

| Flex Tag | Custom Binding |
|---|---|
| ```html<br><div id="cpuAreaChartID"<br>style="height:100%;width:100%"<br>  data-bind="mpChart: {<br>    type: 'area',<br>    timeAxisType: 'enabled',<br>    dataSelection: 'multiple',<br>    emptyText: 'No data',<br>    animationOnDisplay: 'auto',<br>    yAxis: {min: 0, title: '%<br>Utilization'},<br>    targetName:<br>appModel.target.name,<br>    targetType:<br>appModel.target.type,<br>    metricName: 'CPUPerf',<br>    metricColumns: ['non_nice',<br>'nice', 'system', 'io_wait',<br>'irq'],<br>    timePeriod: 'REALTIME',<br>    interval: 30}" ><br></div><br>``` | ```html<br><mp-chart id="cpuAreaChartID"<br>style="height:100%;width:100%"<br>  params="mpChart: {<br>    type: 'area',<br>    timeAxisType: 'enabled',<br>    dataSelection: 'multiple',<br>    emptyText: 'No data',<br>    animationOnDisplay: 'auto',<br>    yAxis: {min: 0, title: '%<br>Utilization'},<br>    targetName:<br>appModel.target.name,<br>    targetType:<br>appModel.target.type,<br>    metricName: 'CPUPerf',<br>    metricColumns: ['non_nice',<br>'nice', 'system', 'io_wait',<br>'irq'],<br>    timePeriod: 'REALTIME',<br>    interval: 30}" ><br></mp-chart><br>``` |

## Bar Chart

The column chart differs from the bar chart only with the use of the 'orientation' property. If not specified, it defaults to vertical (or a column chart).

**Using customDataSource:**

| Flex Tag | Custom Binding |
|---|---|
| ```html<br><div id="bchart_from_custom"<br>style="height:100%;width:100%"<br>  data-bind="mpChart: {<br>    type: 'bar',<br>    orientation: 'horizontal',<br>    groupBy: 'byColumn',<br>    customDataSource:<br>model().fsTypeDataSource,<br>    emptyText: 'No data',<br>    animationOnDisplay: 'auto'}" ><br></div><br>``` | ```html<br><mp-chart id="bchart_from_custom"<br>style="height:100%;width:100%"<br>  data-bind="mpChart: {<br>    type: 'bar',<br>    orientation: 'horizontal',<br>    groupBy: 'byColumn',<br>    customDataSource:<br>model().fsTypeDataSource,<br>    emptyText: 'No data',<br>    animationOnDisplay: 'auto'}"><br></mp-chart><br>``` |

from controller.init:

```
var fsTypeData = page.getModel("fsTypeDataSource");
        if (fsTypeData == null) {
            fsTypeData = new CustomDataSource([new
QueryColumnDesc("nfs", QueryColumnType.STRING),
                                    new
QueryColumnDesc("ext3", QueryColumnType.STRING),
                                    new
QueryColumnDesc("tmpfs", QueryColumnType.STRING)]);
            page.setModel("fsTypeDataSource", fsTypeData);
        }
```

**Using dataService:**

| Flex Tag | Custom Binding |
| --- | --- |
| <div id="bchart_from_sql" style="height:100%;width:100%" data-bind="mpChart: { type: 'bar', orientation: 'horizontal', groupBy: 'byColumn', dataService: 'cpu_usage_from_sql', xAxis: {title: 'CPU Number'}, emptyText: 'No data', animationOnDisplay: 'auto'}"> </div> | <mp-chart id="bchart_from_sql" style="height:100%;width:100%" params="mpChart: { type: 'bar', orientation: 'horizontal', groupBy: 'byColumn', dataService: 'cpu_usage_from_sql', xAxis: {title: 'CPU Number'}, emptyText: 'No data', animationOnDisplay: 'auto'}"> </mp-chart> |
| <div id="cpu_usage_from_sql" data-bind="mpSqlDataService : { queryID:'CPU_USAGE', properties:props('TARGET_GUID',app Model.target.guid) }"> </div> | |

## Column Chart

This is the start of your topic.The column chart differs from the bar chart only with the use of the 'orientation' property. If not specified, it defaults to vertical (or a column chart).

**Using dataService:**

| Flex Tag | Custom Binding |
|---|---|
| ```
<div id="bchart_from_custom2"
style="height:100%;width:100%"
  data-bind="mpChart: {
    type: 'bar',
    dataService: 'topFiveFsData',
    emptyText: 'No data',
    animationOnDisplay: 'auto'}" >
</div>
``` | ```
<mp-chart
id="bchart_from_custom2"
style="height:100%;width:100%"
  params="mpChart: {
    type: 'bar',
    dataService: 'topFiveFsData',
    emptyText: 'No data',
    animationOnDisplay: 'auto'}">
</mp-chart>
``` |

```
<mp-metric-values-data-service id="topFiveFsData"
        params="metricName:'FilesystemPerf',
          columns:['Utilization'],
          targetName:appModel.target.name,
          targetType:appModel.target.type,
          timePeriod:'REALTIME',
          interval: 30">
      </mp-metric-values-data-service>
```

**Using metric specification:**

| Flex Tag | Custom Binding |
|---|---|
| ```
<div id="processorChart"
style="height:100%;width:100%"
  data-bind="mpChart: {
    type: 'bar',
    orientation: 'vertical',
    dataSelection: 'multiple',
    emptyText: 'No data',
    animationOnDisplay: 'auto',
    targetName:
appModel.target.name,
    targetType:
appModel.target.type,
    metricName:
'CPUProcessorPerf',
    metricColumns: ['CPUIdle'],
    timePeriod: 'LAST_DAY'}" >
</div>
``` | ```
 <mp-chart id="processorChart"
style="height:100%;width:100%"
  params="mpChart: {
    type: 'bar',
    orientation: 'vertical',
    dataSelection: 'multiple',
    emptyText: 'No data',
    animationOnDisplay: 'auto',
    targetName:
appModel.target.name,
    targetType:
appModel.target.type,
    metricName:
'CPUProcessorPerf',
    metricColumns: ['CPUIdle'],
    timePeriod: 'LAST_DAY'}" >
</mp-chart>
``` |

## Pie Chart

**Using customDataSource:**

| Flex Tag | Custom Binding |
|---|---|
| ```<div id="cpuByUserChart" style="height:100%;width:100%"   data-bind="mpChart: {     type: 'pie',     title: {text: 'CPU % per User'},     customDataSource: model().userSummaryData,     selectedColumn: model().userCpuCol,     emptyText: 'No data',     animationOnDisplay: 'auto'}" > </div>``` | ```<mp-chart id="cpuByUserChart" style="height:100%;width:100%"   params="mpChart: {     type: 'pie',     title: {text: 'CPU % per User'},     customDataSource: model().userSummaryData,     selectedColumn: model().userCpuCol,     emptyText: 'No data',     animationOnDisplay: 'auto'}" > </mp-chart>``` |

from `controller.init`:

```
var userSummaryData = new CustomDataSource(["User","CPU %","Process Count"], true);
        page.setModel("userSummaryData", userSummaryData);
```

**Using dataService:**

| Flex Tag | Custom Binding |
|---|---|
| ```<div id="pchart_from_mvds" style="height:100%;width:100%"   data-bind="mpChart: {     type: 'pie',     dataService: 'memory_data_from_metric_realtime' ,     emptyText: 'No data',     animationOnDisplay: 'auto'}"> </div>``` | ```<mp-chart id="pchart_from_mvds" style="height:100%;width:100%"   params="mpChart: {     type: 'pie',     dataService: 'memory_data_from_metric_realtime' ,     emptyText: 'No data',     animationOnDisplay: 'auto'}"> </mp-chart>``` |

```
<mp-metric-values-data-service id="memory_data_from_metric_realtime"
        params="flattenData: true,
          metricName:'MemoryPerf',
          columns:
['MemFree','Buffers','Cached','SwapCached','Active','Inactive'],
          targetName:appModel.target.name,
          targetType:appModel.target.type,
          timePeriod:'REALTIME',
          interval: 15 ">
      </mp-metric-values-data-service>
```

**ORACLE**

**Using metric specification:**

| Flex Tag | Custom Binding |
|---|---|
| ```<div id="memChart"``` <br> ```style="height:80%;width:100%"``` <br> ```  data-bind="mpChart: {``` <br> ```    type: 'pie',``` <br> ```    dataSelection: 'multiple',``` <br> ```    styleDefaults: {colors:``` <br> ```Colors.DEFAULT_COLORS},``` <br> ```    targetName:``` <br> ```appModel.target.name,``` <br> ```    targetType:``` <br> ```appModel.target.type,``` <br> ```    metricName: 'MemoryPerf',``` <br> ```    metricColumns:``` <br> ```model().memChartColumns,``` <br> ```    timePeriod: 'CURRENT',``` <br> ```emptyText: 'No data',``` <br> ```    animationOnDisplay: 'auto'}"``` <br> ```>``` <br> ```</div>``` | ```<mp-chart id="memChart"``` <br> ```style="height:80%;width:100%"``` <br> ```  params="mpChart: {``` <br> ```    type: 'pie',``` <br> ```    dataSelection: 'multiple',``` <br> ```    styleDefaults: {colors:``` <br> ```Colors.DEFAULT_COLORS},``` <br> ```    targetName:``` <br> ```appModel.target.name,``` <br> ```    targetType:``` <br> ```appModel.target.type,``` <br> ```    metricName: 'MemoryPerf',``` <br> ```    metricColumns:``` <br> ```model().memChartColumns,``` <br> ```    timePeriod: 'CURRENT',``` <br> ```emptyText: 'No data',``` <br> ```    animationOnDisplay: 'auto'}"``` <br> ```>``` <br> ```</mp-chart>``` |

# Prepackaged Regions

## Target Instance Regions

### Incident Region

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| ```<mp:IncidentRegion``` <br> ```id="eventsRegion"``` <br> ```  height="34%"``` <br> ```width="100%" />``` | ```<div data-``` <br> ```bind="mpIncidentRegion``` <br> ```"``` <br><br> ```style="height:33%"></``` <br> ```div>``` | ```<mp-incident-region``` <br> ```style="height:33%">``` <br> ```</mp-incident-region>``` |

### Job Activity Region

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:JobSummaryRegion id="jobSummary" width="100%" height="30%" />` | `<div data-bind="mpJobSummaryRegion" style="height:30%"></div>` | `<mp-job-summary-region style="height:35%;width:100%"> </mp-job-summary-region>` |

**Availability Region**

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:AvailabilityDetails id="availDetails" width="100%" height="100%" daySpan="1"/>` | `<div data-bind="mpAvailabilityRegion" style="height:100%;width:100%"></div>` | `<mp-availability-region style="height:100%;width:100%"> </mp-availability-region>` |

**CredentialsRegion**

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:CredentialsRegion id="sampleCreds" width="40%" height="100%"` `credentialState="{model.sampCredState}"` `credentialType="{'SampleCredsType'}" />` | `<div id="sampleCreds" data-bind="mpCredentialsRegion: { credentialType:'SampleCredsType', credentialState:model().sampCredState }"></div>` | `<mp-credentials-region id="sampleCreds" params="credentialType:'SampleCredsType', credentialState:model().sampCredState"> </mp-credentials-region>` |

**CredentialsDisplay**

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:CredentialsDisplay id="creds" width="100%" height="100%" credentialTypes="{['HostCreds', 'SampleCredsType']}" credentialState="{model.credState}"/>` | `<div id="creds" data-bind="mpCredentialsDisplay: { credentialTypes: ['HostCreds', 'SampleCredsType'], credentialState:model().credState }"></div>` | `<mp-credentials-display id="creds" params="credentialTypes:['HostCreds', 'SampleCredsType'], credentialState:model().credState" > </mp-credentials-display>` |

## System Regions

### Status Overview Region

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:StatusOverviewRegion id="statusRegion" height="34%" width="100%" />` | `<div data-bind="mpStatusOverviewRegion" style="height:33%"></div>` | `<mp-status-overview-region style="height:33%"> </mp-status-overview-region>` |

### Issues Overview Region

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:IssuesOverviewRegion id="issuesRegion" height="34%" width="100%" />` | `<div data-bind="mpIssuesOverviewRegion" style="height:33%"></div>` | `<mp-issues-overview-region style="height:33%"> </mp-issues-overview-region>` |

### Jobs Activity Region

| Flex Tag | Custom Binding | Custom Element |
|---|---|---|
| `<mp:JobsActivityRegion id="jobActivity" width="100%" height="30%" />` | `<div data-bind="mpJobsActivityRegion" style="height:30%"></div>` | `<mp-jobs-activity-region style="height:35%;width:100%"> </mp-jobs-activity-region>` |

# 10
# Customizing Incident Manager

This chapter describes how to customize the event details page to provide more diagnostic information about the event and to facilitate quicker resolution of the underlying issue. Details pages on the Incident Manager UI allow users to view the details of an event. The content of such pages helps the user understand the basic nature of the underlying issue and provides additional contextual details (such as text, links to diagnostic or resolution pages) to resolve the issue quickly.

For incidents that have only one event, the customizations applied to the event details page are automatically applied to the Incident Details page.

> **Note:**
>
> For information about Incident Management, see the Using Incident Management chapter of the *Oracle Enterprise Manager Cloud Control Administrator's Guide*.

This chapter contains the following sections:

- Introduction to Customizing Incident Manager
- Understanding Supported Customizations
- Creating Event-Specific Customization XML
- Adding Customized Details About the Event
- Providing Content in the Guided Resolution Region
- Defining a Search String for My Oracle Support Knowledge
- Defining Conditions for Customization
- Registering Customizations
- Testing Incident Manager After Customization

## Introduction to Customizing Incident Manager

As a plug-in developer, you are responsible for the following steps within customizing Incident Manager:

1. Determine what customizations you require for your Incident Manager UI. For fine-grained access, use conditions. For more information, see Understanding Supported Customizations.

2. Construct the customization XML according to the XSD. For more information, see Creating Event-Specific Customization XML .

3. Register your customization. For more information, see Registering Customizations.

---

4. Test the UI by publishing an event that matches the condition. For more information, see Testing Incident Manager After Customization.

# Understanding Supported Customizations

Figure 10-1 displays the General tab for a selected incident from the Incident Manager page.

**Figure 10-1    Incident Manager**



The following customizations are supported for this page:

- Adding name-value pairs to the Incident Details region.

  For more information, see Adding Customized Details About the Event.

- Customizing Action and Diagnostic links in the Guided Resolution region.

  For more information, see Providing Content in the Guided Resolution Region .

- Adding recommendations to the Guided Resolution region.

  For more information, see Adding Recommendations using XML .

- Specifying the default search phrase for My Oracle Support Knowledge.

  For more information, see Defining a Search String for My Oracle Support Knowledge.

Each customization specification has two parts:

1. Condition

   This is the criteria used to identify an event for which the customized content will be rendered. For example, consider a scenario where you want to show a

diagnostic link for metric alerts on a database. The condition would be "event class is metric_alert and target type is oracle_database". Another example is where you to show the region containing a metric chart. This condition would be "event class is metric_alert and metric_type is numeric".

> **Note:**
>
> Any target type name is supported. While matching an event, you match the target type in the condition with the target type of the event

2. Action

   The actions specify the customized content. For example, the specification of the diagnostic link (that is, the label and the URL to be shown under it).

# Creating Event-Specific Customization XML

Oracle provides an event-specific customization XSD so that you can write XML to describe customizations for a specific event for display on the Incident Manager UI.

> **Note:**
>
> For a complete event-specific customization XML Schema Definitions (XSD), see the Extensibility Development Kit (EDK).

The event-specific customization XSD defines how the Incident Manager UI supports UI customization.

You can define fine-grained conditions to customize the Event Details or Incident Details pages.

**Example: Sample Metadata File**

```
<evt:CustomUI AppliesTo="EVENT" EventClass ="metric_alert" TargetType =host">
    <evt:ConditionDetails>
       <evt:Condition>
         <evt:Attrib Name="metric_name" Value="Load"/>
         <evt:Attrib Name="metricColumn" Value="cpuUtil"/> </evt:Condition>
     </evt:ConditionDetails>
</evt:CustomUI>
```

Oracle recommends the following naming conventions for your metadata XML:

*   *event_class_description*.xml

    In the preceding file name:

    –   *event_class* represents the name of the event class

        Event customization supports the following event classes:

        *   metric_alert

        *   target_availability

* job_status_change

* cs_rule_violation

* cs_score

* sla_alert

* metric_error

– *description* represents a short description of the event customization

For example, job_status_change_recommendation.xml

- *event_class_target_type_description*.xml

    In the preceding file name:

    – *event_class* represents the name of the event class

    – *target_type* represents the name of the target type for which this event is generated

    – *description* represents a short description of the event customization

    For example, host_metric_alert_diaglinks.xml

> **Note:**
>
> The maximum length of the file name is 255 characters.

For information about the directory location for the metadata XML, see Registering Customizations.

> **Note:**
>
> Use the `empdk validate_plugin` command to validate the XML metadata file. For more information about the `empdk validate_plugin` command, see Validating, Packaging, and Deploying the Plug-in .

# Overview of Event-Specific Customization Metadata Elements

Table 10-1 describes the key elements that define the event-specific customization XML.

**Table 10-1    Key Elements in Event-Specific Customization XML**

| Element | Description |
| --- | --- |
| `evt:CustomUI` | This is the root element of the XML. It defines the customization. |
| | It includes the following attributes: |
| | • `AppliesTo`: Applicable to event customizations. The only valid value is EVENT. |
| | • `EventClass`: Specifies the internal event class name and is applicable only when the customization applies to an event. |
| | • `TargetType`: Internal name of the target type. The customization applies to events from all targets of this target type. |
| `evt:ConditionDetails` | Specifies the criteria on which the customizations are to be applied. |
| `evt:Condition` | Specifies a condition for the customization. |
| | **Note**: Oracle supports one condition only within the `evt:ConditionDetails` tag. |
| `evt:DetailUI` | Specifies that you are customizing the Details region of the Incident Manager UI page. |
| | For more information, see Adding Customized Details About the Event. |
| `evt:GuidedResolutionDetails` | Specifies that you are customizing the Guided Resolution region of the Incident Manager UI page. Using this element, you can add action links, diagnostic links, and recommendations. |
| | For more information, see Adding Recommendations using XML . |

# About Events

This section provides common event attributes and the definition of the two most commonly-used event types:

• Common Event Attributes

• Target Availability Event

• Metric Alert Event

# Common Event Attributes

All events have the following common attributes:

**Table 10-2    Common Event Attributes**

| Attribute | Description |
| --- | --- |
| sys_event_class | Event type |
| | Possible values: |
| | • target_availability: Target Availability events |
| | • metric_alert: Metric Alert events |
| sys_event_name | Event name to identify the nature of the event uniquely |
| sys_event_key | Name of a subcomponent within the event source object to which this event is related. This is optional. Examples include a disk name on a host, name of a tablespace, and so on |
| sys_event_msg | Event message |
| sys_action_msg | Action message |
| sys_source_obj_type | Source object type. For example, JOBS for job-based events. |
| sys_source_obj_id | Unique internal identifier of a Source object |
| sys_target_guid | Unique internal identifier of a target |
| sys_target_name | Target name |
| sys_target_owner | Target owner |
| sys_target_version | Target version |
| sys_target_lifecycle_status | Lifecycle status |
| sys_incident_id | Incident ID |
| sys_severity | Severity of the event |
| | Possible values: |
| | • 32: Fatal |
| | • 16: Critical |
| | • 8: Warning |
| | • 4: Minor Warning |
| sys_category | Event category. |
| | Possible values: |
| | • Availability: 1 |
| | • Configuration: 2 |
| | • Capacity: 4 |
| | • Fault: 8 |
| | • Load: 16 |
| | • Performance: 32 |
| | • Security: 64 |
| | • Jobs: 128 |
| | • Diagnostics: 256 |
| | • Error: 512 |
| | • Business: 1024 |

## Target Availability Event

The Target Availability Event represents a target's availability status.

The following example shows the event attributes defined by the target availability XML file. Table 10-3 provides a list of all the event attributes for target availability.

**Example: target_availability.xml file**

```
<evt:EventClass Name="target_availability"
          NLSID="TARGET_AVAILABILITY"

ResourceBundle="oracle.sysman.core.common.events.classes.rsc.availability.Availab
ilityEventsMsg"
          TargetAware="ALWAYS"
          SourceObjectType="TARGET"
          Version="1.0"
          xmlns:evt="http://www.oracle.com/EnterpriseGridControl/
eventclass_model"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/
eventclass_model EventClass.xsd">

    <evt:DescriptionNLSID>TARGET_AVAILABILITY_DESC</evt:DescriptionNLSID>

    <evt:AttributeDef>

        <!--This attribute is used to store the availability status of a target--
>
        <evt:Attrib Name="target_status"
                    DataType="STRING"
                    isReferenced="false"
                    NLSID="TARGET_STATUS"
                    isValueTranslatable="true">
            <evt:DescriptionNLSID>TARGET_STATUS_DESC</evt:DescriptionNLSID>
        </evt:Attrib>

        <!--The guid of the severity record associated with this availability
record-->
        <evt:Attrib Name="severity_guid"
                    DataType="RAW"
                    isReferenced="false"
                    NLSID="SEVERITY_GUID"
                    isValueTranslatable="false">
            <evt:DescriptionNLSID>SEVERITY_GUID_DESC</evt:DescriptionNLSID>
        </evt:Attrib>

        <!--The cycle guid of the severity record associated with this
availability record-->
        <evt:Attrib Name="cycle_guid"
                    DataType="RAW"
                    isReferenced="true"
                    NLSID="CYCLE_GUID"
                    isValueTranslatable="false">
            <evt:DescriptionNLSID>CYCLE_GUID_DESC</evt:DescriptionNLSID>
        </evt:Attrib>

        <!--The below attributes specifies the metric guid of response metric -->
        <evt:Attrib Name="metric_guid"
                    DataType="RAW"
                    isReferenced="true"
                    NLSID="METRIC_GUID"
                    isValueTranslatable="false">
            <evt:DescriptionNLSID>METRIC_GUID_DESC</evt:DescriptionNLSID>
        </evt:Attrib>

        <!--The below attribute represents a sub-state for availability states
like Status pending, Agent Unreachable and Blackout.
```

```
        TARGET STATUS         CODE     SUB_STATE

        Any                   0        None (Default)
        Agent unreachable     1        Normal
        Agent unreachable     2        Host Down
        Agent unreachable     3        Disk Full
        Status Pending        10       Normal
        Status Pending        11       Stuck
        -->
        <evt:Attrib Name="avail_sub_state"
                   DataType="NUMBER"
                   isReferenced="false"
                   NLSID="AVAILABILITY_SUB_STATE"
                   isValueTranslatable="false">
            <evt:DescriptionNLSID>AVAILABILITY_SUB_STATE_DESC</
evt:DescriptionNLSID>
        </evt:Attrib>

        <!--The below attributes specifies the availability transition severity
        that resulted in the target availability status that is specified by
        target_status attribute -->
        <evt:Attrib Name="avail_severity"
                   DataType="NUMBER"
                   isReferenced="false"
                   NLSID="AVAILABILITY_SEVERITY"
                   isValueTranslatable="false">
            <evt:DescriptionNLSID>AVAILABILITY_SEVERITY_DESC</
evt:DescriptionNLSID>
        </evt:Attrib>
    </evt:AttributeDef>

    <evt:RefAttribSource><![CDATA[ mgmt_avail.get_target_avail_ref_attribs]]></
evt:RefAttribSource>

    <!-- For availibility we don't have any identifier attribute list. -->
    <!-- So system will use target_guid, event_class name to generate the
identifier attribute. -->

    <evt:RuleAttribs>
        <evt:RuleAttrib Name="target_status" /></evt:RuleAttrib>
        <evt:RuleAttrib Name="avail_sub_state" /></evt:RuleAttrib>
        <evt:RuleAttrib Name="avail_severity" /></evt:RuleAttrib>
    </evt:RuleAttribs>

    <evt:NotifAttribs>
        <evt:NotifAttrib Name="target_status" />
        <evt:NotifAttrib Name="severity_guid" />
        <evt:NotifAttrib Name="avail_sub_state" />
        <evt:NotifAttrib Name="avail_severity" />
        <evt:NotifAttrib Name="metric_guid" />
        <evt:NotifAttrib Name="cycle_guid" />
    </evt:NotifAttribs>

    <evt:Severities>
        <evt:Severity>FATAL</evt:Severity>
        <evt:Severity>CRITICAL</evt:Severity>
        <evt:Severity>WARNING</evt:Severity>
        <evt:Severity>MINOR_WARNING</evt:Severity>
        <evt:Severity>INFORMATIONAL</evt:Severity>
    </evt:Severities>
</evt:EventClass>
```

**Table 10-3    Event Attributes for Target Availability**

| Attribute | Description |
| --- | --- |
| TARGET_STATUS | Availability status |
| AVAILABILITY_SUB_STATE | Availability substatus |
| AVAILABILITY_SEVERITY | Transition severity |

## Metric Alert Event

A metric alert event is generated when an alert occurs for a metric on a specific target (for example, CPU utilization for a host target) or metric on a target and object combination (for example, space usage on a specific tablespace of a database target)

The following example shows the event attributes defined by the metric alert XML file. Table 10-4 provides a list of all the event attributes for the metric alert event.

**Example: metric_alert.xml**

```
<evt:EventClass Name="metric_alert"
          NLSID="METRIC_ALERT_EVENT"
          TargetAware="ALWAYS"
          SourceObjectType="TARGET"

ResourceBundle="oracle.sysman.core.common.events.classes.rsc.metrics.MetricEvents
Msg"
          Version="1.1"
          xmlns:evt="http://www.oracle.com/EnterpriseGridControl/
eventclass_model"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/
eventclass_model EventClass.xsd">
    <evt:DescriptionNLSID>METRIC_ALERT_DESC</evt:DescriptionNLSID>
    <evt:AttributeDef>
        <evt:Attrib Name="metric_guid" DataType="RAW" isReferenced="false"
                    NLSID="METRIC_GUID_NLSID" isValueTranslatable="false">
            <evt:DescriptionNLSID>METRIC_GUID_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_value" DataType="STRING" isReferenced="false"
                    NLSID="KEY_VALUE_NLSID" isValueTranslatable="false">
            <evt:DescriptionNLSID>KEY_VALUE_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="severity_guid" DataType="RAW" isReferenced="false"
                    NLSID="SEVERITY_GUID_NLSID" isValueTranslatable="false">
            <evt:DescriptionNLSID>SEVERITY_GUID_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="cycle_guid" DataType="RAW" isReferenced="true"
                    NLSID="CYCLE_GUID_NLSID" isValueTranslatable="false">
            <evt:DescriptionNLSID>CYCLE_GUID_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="value" DataType="STRING" isReferenced="true"
                    NLSID="VALUE_NLSID" isValueTranslatable="false">
            <evt:DescriptionNLSID>VALUE_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="metric_group" DataType="STRING" isReferenced="true"
                    NLSID="METRIC_GROUP_NLSID" isValueTranslatable="true">
            <evt:DescriptionNLSID>METRIC_GROUP_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
```

```
        <evt:Attrib Name="metric_column" DataType="STRING" isReferenced="true"
                    NLSID="METRIC_COLUMN_NLSID" isValueTranslatable="true">
            <evt:DescriptionNLSID>METRIC_COLUMN_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="metric_description" DataType="STRING"
isReferenced="true"
                    NLSID="METRIC_DESCRIPTION_NLSID" isValueTranslatable="true">
            <evt:DescriptionNLSID>METRIC_DESCRIPTION_DESC_NLID</
evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="coll_name" DataType="STRING" isReferenced="true"
                    NLSID="COLL_NAME_NLSID" isValueTranslatable="true">
            <evt:DescriptionNLSID>COLL_NAME_DESC</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_1" DataType="STRING" isReferenced="true"
                    NLSID="ALERT_KEY_COL_NLSID_1" isValueTranslatable="true">
            <evt:DescriptionNLSID>ALERT_KEY_COL_DESC_1</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_2" DataType="STRING" isReferenced="true"
                    NLSID="ALERT_KEY_COL_NLSID_2" isValueTranslatable="true">
            <evt:DescriptionNLSID>ALERT_KEY_COL_DESC_2</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_3" DataType="STRING" isReferenced="true"
                    NLSID="ALERT_KEY_COL_NLSID_3" isValueTranslatable="true">
            <evt:DescriptionNLSID>ALERT_KEY_COL_DESC_3</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_4" DataType="STRING" isReferenced="true"
                    NLSID="ALERT_KEY_COL_NLSID_4" isValueTranslatable="true">
            <evt:DescriptionNLSID>ALERT_KEY_COL_DESC_4</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_5" DataType="STRING" isReferenced="true"
                    NLSID="ALERT_KEY_COL_NLSID_5" isValueTranslatable="true">
            <evt:DescriptionNLSID>ALERT_KEY_COL_DESC_5</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_6" DataType="STRING" isReferenced="true"
                    NLSID="ALERT_KEY_COL_NLSID_6" isValueTranslatable="true">
            <evt:DescriptionNLSID>ALERT_KEY_COL_DESC_6</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_7" DataType="STRING" isReferenced="true"
                    NLSID="ALERT_KEY_COL_NLSID_7" isValueTranslatable="true">
            <evt:DescriptionNLSID>ALERT_KEY_COL_DESC_7</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_1_value" DataType="STRING"
isReferenced="true"
                    NLSID="KEY_VALUE_PART_NLSID_1" isValueTranslatable="false">
            <evt:DescriptionNLSID>KEY_VALUE_PART_DESC_1</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_2_value" DataType="STRING"
isReferenced="true"
                    NLSID="KEY_VALUE_PART_NLSID_2" isValueTranslatable="false">
            <evt:DescriptionNLSID>KEY_VALUE_PART_DESC_2</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_3_value" DataType="STRING"
isReferenced="true"
                    NLSID="KEY_VALUE_PART_NLSID_3" isValueTranslatable="false">
            <evt:DescriptionNLSID>KEY_VALUE_PART_DESC_3</evt:DescriptionNLSID>
        </evt:Attrib>
        <evt:Attrib Name="key_column_4_value" DataType="STRING"
isReferenced="true"
                    NLSID="KEY_VALUE_PART_NLSID_4" isValueTranslatable="false">
            <evt:DescriptionNLSID>KEY_VALUE_PART_DESC_4</evt:DescriptionNLSID>
```

```
            </evt:Attrib>
            <evt:Attrib Name="key_column_5_value" DataType="STRING"
isReferenced="true"
                        NLSID="KEY_VALUE_PART_NLSID_5" isValueTranslatable="false">
                <evt:DescriptionNLSID>KEY_VALUE_PART_DESC_5</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="key_column_6_value" DataType="STRING"
isReferenced="true"
                        NLSID="KEY_VALUE_PART_NLSID_6" isValueTranslatable="false">
                <evt:DescriptionNLSID>KEY_VALUE_PART_DESC_6</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="key_column_7_value" DataType="STRING"
isReferenced="true"
                        NLSID="KEY_VALUE_PART_NLSID_7" isValueTranslatable="false">
                <evt:DescriptionNLSID>KEY_VALUE_PART_DESC_7</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="metric_type" DataType="NUMBER" isReferenced="true"
                        NLSID="METRIC_TYPE" isValueTranslatable="false">
                <evt:DescriptionNLSID>METRIC_TYPE_DESC</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="num_keys" DataType="NUMBER" isReferenced="true"
                        NLSID="NUM_KEYS" isValueTranslatable="false">
                <evt:DescriptionNLSID>NUM_KEYS_DESC</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="unit" DataType="STRING" isReferenced="true"
                        NLSID="UNIT_NLSID" isValueTranslatable="true">
                <evt:DescriptionNLSID>UNIT_DESC</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="is_thresholdable" DataType="NUMBER"
isReferenced="true"
                        NLSID="IS_THRESHOLDABLE" isValueTranslatable="false">
                <evt:DescriptionNLSID>IS_THRESHOLDABLE_DESC</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="is_remote" DataType="NUMBER" isReferenced="true"
                        NLSID="IS_REMOTE" isValueTranslatable="false">
                <evt:DescriptionNLSID>IS_REMOTE_DESC</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="is_long_running" DataType="NUMBER" isReferenced="true"
                        NLSID="IS_LONG_RUNNING" isValueTranslatable="false">
                <evt:DescriptionNLSID>IS_LONG_RUNNING_DESC</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="is_udm" DataType="NUMBER" isReferenced="true"
                        NLSID="IS_UDM" isValueTranslatable="false">
                <evt:DescriptionNLSID>IS_UDM_DESC</evt:DescriptionNLSID>
            </evt:Attrib>
            <evt:Attrib Name="is_metric_extension" DataType="NUMBER"
isReferenced="true"
                        NLSID="IS_METRIC_EXTENSION" isValueTranslatable="false">
                <evt:DescriptionNLSID>IS_METRIC_EXTENSION_DESC</
evt:DescriptionNLSID>
            </evt:Attrib>
    </evt:AttributeDef>

    <evt:RefAttribSource><!
[CDATA[sysman.em_severity.get_metric_alert_ref_attribs]]></evt:RefAttribSource>

    <evt:SignatureAttribs>
        <evt:SignaturePart>metric_guid</evt:SignaturePart>
        <evt:SignaturePart>key_value</evt:SignaturePart>
    </evt:SignatureAttribs>
```

```
            <evt:RuleAttribs>
                <evt:RuleAttrib Name="metric_group"/>
                <evt:RuleAttrib Name="metric_column"/>
                <evt:RuleAttrib Name="key_value"/>
                <evt:RuleAttrib Name="key_column_1_value"/>
                <evt:RuleAttrib Name="key_column_2_value"/>
                <evt:RuleAttrib Name="key_column_3_value"/>
                <evt:RuleAttrib Name="key_column_4_value"/>
                <evt:RuleAttrib Name="key_column_5_value"/>
                <evt:RuleAttrib Name="key_column_6_value"/>
                <evt:RuleAttrib Name="key_column_7_value"/>
            </evt:RuleAttribs>

            <evt:NotifAttribs>
                <evt:NotifAttrib Name="metric_guid"/>
                <evt:NotifAttrib Name="severity_guid"/>
                <evt:NotifAttrib Name="cycle_guid"/>
                <evt:NotifAttrib Name="coll_name"/>
                <evt:NotifAttrib Name="metric_group"/>
                <evt:NotifAttrib Name="metric_column"/>
                <evt:NotifAttrib Name="metric_description"/>
                <evt:NotifAttrib Name="value"/>
                <evt:NotifAttrib Name="key_value"/>
                <evt:NotifAttrib Name="key_column_1"/>
                <evt:NotifAttrib Name="key_column_1_value"/>
                <evt:NotifAttrib Name="key_column_2"/>
                <evt:NotifAttrib Name="key_column_2_value"/>
                <evt:NotifAttrib Name="key_column_3"/>
                <evt:NotifAttrib Name="key_column_3_value"/>
                <evt:NotifAttrib Name="key_column_4"/>
                <evt:NotifAttrib Name="key_column_4_value"/>
                <evt:NotifAttrib Name="key_column_5"/>
                <evt:NotifAttrib Name="key_column_5_value"/>
                <evt:NotifAttrib Name="key_column_6"/>
                <evt:NotifAttrib Name="key_column_6_value"/>
                <evt:NotifAttrib Name="key_column_7"/>
                <evt:NotifAttrib Name="key_column_7_value"/>
                <evt:NotifAttrib Name="num_keys"/>
            </evt:NotifAttribs>

            <evt:Severities>
                <evt:Severity>CRITICAL</evt:Severity>
                <evt:Severity>WARNING</evt:Severity>
            </evt:Severities>

</evt:EventClass>
```

**Table 10-4    Event Class Attributes for Metric Alerts**

| Attribute | Description |
| --- | --- |
| KEY_VALUE_DESC | Monitored object for the metric corresponding to the Metric Alert event |
| VALUE_DESC | Value of the metric when the event triggered |
| METRIC_GROUP_DESC | The name of the metric |
| METRIC_COLUMN_DESC | The name of the metric column |
| KEY_COLUMN_1_VALUE | Value of Key Column 1 |
| KEY_COLUMN_2_VALUE | Value of Key Column 2 |

**Table 10-4    (Cont.) Event Class Attributes for Metric Alerts**

| Attribute | Description |
|---|---|
| KEY_COLUMN_3_VALUE | Value of Key Column 3 |
| KEY_COLUMN_4_VALUE | Value of Key Column 4 |
| KEY_COLUMN_5_VALUE | Value of Key Column 5 |
| KEY_COLUMN_6_VALUE | Value of Key Column 6 |
| KEY__COLUMN_7_VALUE | Value of Key Column 7 |
| IS_METRIC_EXTENSION_ DESC | Flag to indicate if the metric is metric extension |

# Adding Customized Details About the Event

The Incident Details region shows information about the event. It consists of system attributes (such as the message, target name, and when the event was reported) and the class attributes. You can customize the name-value pairs for the class attributes.

Through the event-specific customization XML, you can choose which attributes to show, such as the labels for the name part, and whether you require a link under the value. For more information, see Creating Event-Specific Customization XML .

**Example: Constructing a Name-Value Pair**

```
<evt:DetailUI>
  <evt:UIAttributeList>
    <evt:UIAttrib Name="metric_name">
      <evt:URL PageType="sdkcore-dummy-published-page-id">
      <evt:URLParam Name="target" Value="^TARGET:sys_target_name^"/>
      <evt:URLParam Name="type" Value="^TARGET:sys_target_type^"/>
      <evt:URLParam Name="metric" Value="^metric_name^"/>
      <evt:URLParam Name="metricColumn" Value="^metric_column^"/>
      <evt:URLParam Name="ctxType" Value="Hosts"/>
      </evt:URL>
    </evt:UIAttrib>
  </evt:UIAttributeList>
</evt:DetailUI>
```

The previous example constructs a name-value pair under the Incident Details region. The name is the translated value of metric_name, which is an event class attribute. The value part is the value of metric_name, with a link to the METRIC_DETAILS page with specified URL parameters.

> **Note:**
>
> For the `evt:URL` tag, you must use an EDK published page id as the pageType. At design-time, you cannot validate the link navigation so if you are using this API, then you must verify that the link works correctly on the Incident Manager UI.

The URL parameters in the links can be:

- Event attributes: Must be enclosed in carets (^). For example, ^metric_name^.

  For information about event attributes, see About Events.

- Target attributes: Must be prefixed with TARGET. For example,
  ^TARGET:sys_target_type^.

  Possible target attributes:

  – sys_target_name

  – sys_target_type

  – sys_target_owner

  – sys_target_version

  – sys_target_lifecycle_status

  – sys_target_guid

    For more information about these attributes, see Table 10-2.

- Event context attributes: Must be prefixed with EVENT. For example,
  ^EVENT:evt_context_attrib_name^.

  Event context attributes enable the event publisher to provide additional event
  information to event attributes and are defined as name-value pairs.

- Constants: Must be specified as literal strings. For example, byDay.

> ✎ **Note:**
>
> You can customize the Incident Details region to include class-specific
> attributes only.

# Providing Content in the Guided Resolution Region

You can make the following customizations to the Guided Resolution region:

1. Customize Repair and Diagnostic links (these links can be added or removed)

2. Add recommendations to the Guided Resolution region

3. Specify a default search phrase for My Oracle Support Knowledge

4. Add areas with text to the Guided Resolution region

The Guided Resolution region provides links to relevant Enterprise Manager pages to
help users debug and resolve issues. These context-sensitive links are grouped into
multiple areas based on the nature of content in the destination page of the link. The
following areas are displayed only if they have content.

- Diagnose: This area contains links that can help users diagnose the issue. For
  example, for an event based on a target, the Diagnose area contains a link called
  **View topology** that drills down to the Topology Viewer. You can add or remove
  links in this area using the `evt:DiagLinks` tag (see the Adding a Link to the
  Diagnostics Subsection example).

- Repair: This area contains links that can help users resolve the issue. For
  example, for metric alerts with thresholds, this area might contain a link to **Edit**

**Thresholds**. You can add or remove links to and from this area using the `evt:ActionLinks` tag. This area displays for open events only.

- Recommendation: This area contains text content drawn from the metric advisory, if available. You can customize the content of this area using the `evt:Recommendation` tag (see the Adding Recommendations example in Adding Recommendations using XML ).

- Any additional area: This area contains text content drawn from the metric advisory, if available. You can customize the content of this section using the `evt:Sections` tag (see the Adding Customized Areas example in Customizing Sections).

The following example provides an example of adding a link to the Diagnostics list in the Guided Resolution region using XML:

**Example: Adding a Link to the Diagnostics Subsection**

```
<evt:DiagLinks>
  <evt:Add>
    <evt:Link LinkID="diagLink1_example">
      <evt:Label>
      <evt:LocalizedLabel DefaultLabel="Edit Thresholds"
        NLSID="EDIT_METRIC_THRESHOLDS"
        ResourceBundle="oracle.sysman.resources.MntrResourceBundle"/>
      </evt:Label>
      <evt:URL PageType="sdkcore-published-page-id-for-edit-threshold">
      <evt:URLParam Name="target" Value="^TARGET:sys_target_name^"/>
      <evt:URLParam Name="type" Value="^TARGET:sys_target_type^"/>
      <evt:URLParam Name="event" Value="doEditTreshold"/>
      <evt:URLParam Name="metric" Value="^metric_name^"/>
      <evt:URLParam Name="collName" Value="^coll_name^"/>
      <evt:URLParam Name="keyValue" Value="^key_value^"/>
      <evt:URLParam Name="metricColumn" Value="^column^"/>
      </evt:URL>
    </evt:Link>
  </evt:Add>
</evt:DiagLinks>
```

In the previous example, the link text is derived from the `evt:Label` specification and the URL is derived from `evt:URL`.

The URL parameters in the links can be:

- Event attributes: Must be enclosed in carets (^). For example, ^metric_name^.

  For information about event attributes, see About Events.

- Target attributes: Must be prefixed with TARGET. For example, ^TARGET:sys_target_type^.

  Possible target attributes:

  - sys_target_name

  - sys_target_type

  - sys_target_owner

  - sys_target_version

  - sys_target_lifecycle_status

  - sys_target_guid

For more information about these attributes, see Table 10-2.

- Event context attributes: Must be prefixed with EVENT. For example, ^EVENT:evt_context_attrib_name^.

  Event context attributes enable the event publisher to provide additional event information to event attributes and are defined as name-value pairs.

- Constants: Must be specified as literal strings. For example, byDay.

# Adding Recommendations using XML

The Actions list in the Guided Resolution region enables you to provide some text describing recommended steps that users can follow to diagnose or resolve the issue. Use any of the Label tags to specify the recommendations.

**Example: Adding Recommendations**

```
<evt:GuidedResolutionDetails>
  <evt:GuidedResolution>
    <evt:Recommendation ID="reco_foo">
      <evt:Label>
      <evt:LocalizedLabel
        DefaultLabel="Recommendation for my event class"
        NLSID="MY_EVENT_CLASS_NLSID"
        ResourceBundle="oracle.sysman.MyResourceBundle"/>
      </evt:Label>
    </evt:Recommendation>
  </evt:GuidedResolution>
</evt:GuidedResolutionDetails>
```

You can use other variants of the `evt:Label` tag to construct more complex Recommendations. For example, to substitute the value of an event attribute, such as alertAction as the recommendation, use the following example:

**Example: Adding a Complex Recommendation**

```
<evt:GuidedResolutionDetails>
  <evt:GuidedResolution>

    <evt:Recommendation ID="reco_unique_id">
      <evt:Label>
        <evt:AttributeValue Name="alertAction" />
      </evt:Label>
    </evt:Recommendation>

  </evt:GuidedResolution>
</evt:GuidedResolutionDetails>
```

# Customizing Sections

You can display textual information by adding sections to the Guided Resolution region. For example, while prioritizing an incident about an out-of-the-box Configuration Standard, it might be useful to see the rationale explaining why that Configuration Standard was added. Each added area has a header and some textual content.

To add sections to the Guided Resolution region, you can add specifications to the customization XML similar to the following example:

**ORACLE®**

**Example: Adding Customized Areas**

```
<evt:Sections>
  <evt:Add>
    <evt:Section ID = "section_eventclass_1">
    <evt:SectionHeader>
      <evt:Label>
        <evt:LocalizedLabel DefaultLabel="Section Hdr 1"

NLSID="TRANSLATION_ID"ResourceBundle="oracle.sysman.MyResourceBundle"/>

      </evt:Label>
    </evt:SectionHeader>
    <evt:SectionText>
      <evt:Label>
        <evt:LocalizedLabel DefaultLabel="Section for my event class"
          NLSID="MY_EVENT_CLASS_NLSID"
ResourceBundle="oracle.sysman.MyResourceBundle"/>
      </evt:Label>
    </evt:SectionText>
    </evt:Section>
  </evt:Add>
</evt:Sections>
```

# Defining a Search String for My Oracle Support Knowledge

The customization framework provides a default search phrase in the following order of preference:

1. Customized value from the plug-in developer

2. ORA error found in the Event Summary

3. Event Summary

The search string can be specified explicitly by using XML as shown in the following example. This search string searches for a metric alert indicating high CPU usage for the plug-in component.

**Example: Defining a Search String**

```
<evt:GuidedResolutionDetails>
  <evt:GuidedResolution>
    <evt:SearchPhrase>High CPU Utilization </evt:SearchPhrase>
  </evt:GuidedResolution>
</evt:GuidedResolutionDetails>
```

# Defining Conditions for Customization

To select the events for which customizations are to be applied, you can define conditions using the attributes of an event. To define conditions using attributes from the event payload (for example, system attributes such as target name, target type, or event-class specific attributes such as metric_name for metric alerts, or event-context attributes), use XML.

Conditions can be defined under the `evt:ConditionDetails` tag. You can specify various event attributes here and they are joined together implicitly using an AND condition.

> **✐ Note:**
>
> You can define one condition only under the `evt:ConditionDetails` tag.

**Example: Defining a Condition**

```
<evt:CustomUI AppliesTo ="EVENT" EventClass ="metric_alert" TargetType =host">
  <evt:ConditionDetails>
    <evt:Condition>
      <evt:Attrib Name="metric_name" Value="Load"/>
      <evt:Attrib Name="metricColumn" Value="cpuUtil"/>

    </evt:Condition>
  </evt:ConditionDetails>
...
</evt:CustomUI>
```

The following operators are supported to define conditions:

- EQ: Equals. This is the default operator
- NE: Not equals
- ISNULL: Is null
- ISNOTNULL: Is not null
- CONTAINS: Can be contained in the string (as a substring)
- BEGINSWITH: Begins with (for example, the event name begins with Tablespace)
- IN: In a predefined set of values (separated by pre-defined delimiter comma (,))
- NOT_IN: Not in a predefined set of values. Use for exclusion

# Registering Customizations

The event-specific customization XML files are located in the *$PLUGIN_ORACLE_HOME*/sysman/metadata/events/custmzn directory. In the shipped version of the product, these XMLs are registered as part of the plug-in installation.

If you are creating the XMLs for the first time in a view that is already set up or to test changes, use the Metadata Registration Service (MRS) to register XML for the event class. For more information about the MRS, see Updating Deployed Metadata Files Using the Metadata Registration Service (MRS).

```
emctl register oms metadata -service eventSpecificCustmzn -file XML filename
-pluginId plugin_name -sysman_pwd sysman -debug
```

For example:

```
emctl register oms metadata -service eventSpecificCustmzn -file
metric_alert_host_load.xml -pluginId test.demo.xyz -sysman_pwd sysman -debug
```

You must restart the Oracle Management Service (OMS) after registering the event-specific customization XML, using the `emctl` command from the OMS home directory (*OMS_HOME*).

```
OMS_HOME>emctl stop oms
OMS_HOME>emctl start oms
```

You might encounter the following errors when you register your event-specific customization XML:

* Syntax error in the XML

  For information about the correct syntax and an example of the XML, see Creating Event-Specific Customization XML .

* Incorrect values for attribute names

  For information about attribute names, see About Events.

* Incorrect credentials

  Incorrect credentials will not allow you to connect to the Management Repository. Ensure that you are using authorized credentials.

## Testing Incident Manager After Customization

Test the Incident Manager UI by publishing an event that matches the condition and then make sure that there is an incident created for it:

1. To access Incident Manager, from the **Enterprise** menu, select **Monitoring**, then select **Incident Manager**.

   The **Incident Manager: My open incidents and problems** page appears.

2. From **Views**, select either of the following:

   * **All open incidents** to find the incident
   * **Events without incidents** if you did not create an incident

# 11

# Using Derived Associations

Effective management of IT infrastructure requires knowledge of the relationships between IT entities. Best practices such as those described by ITIL (Information Technology Infrastructure Library) rely on capturing and using such relationships.

Enterprise Manager Cloud Control 13*c* extends the kinds of relationships being supported and adds a declarative mechanism by which these relationships can be maintained. It also determines the membership of entities in a system based on relationships. Based on accurate relationships, various Enterprise Manager applications and components can support customer uses such as:

- Dependency analysis.

  For instance, to understand the impact (to applications and infrastructure) of shutting down a host.

- Topology viewer.

- Change management.

  For instance, tracking the source of cloned databases.

- End-to-end performance analysis, in which interdependencies between application components must be known in order to analyze and isolate issues.

- Change tracking of relationships, such as changes in the way VM resources are allocated.

This chapter covers the following:

- Introduction to Derived Associations

- Understanding Enterprise Manager Association Concepts

- About Association Derivation Rules Management

- Ensuring Performance

- Using Overlapping Associations

- Frequently Asked Questions

## Introduction to Derived Associations

As an plug-in developer, you are responsible for defining those association types that apply to your managed entity types and for verifying that the correct associations (association instances) are present.

A manageable entity is an entity that Enterprise Manager is capable of managing. This implies that the entity is exposed in some form to end users in the Cloud Control application, and has well-defined attributes and semantics.

As a plug-in developer, you are responsible for the following steps with regard to derived associations:

1. Identify all associations that need to be represented for any managed entities (MEs) that you own.

   This generally includes any containment or dependency associations between an ME you own and any other MEs. For each kind of association identified, you may need to coordinate with the owner of the related ME type to determine who should be responsible for assuring that association instances of that type are kept up to date. Some associations (in particular, `hosted_by` and `managed_by`) are automatically maintained by Enterprise Manager, so association derivation rules should not be used for these.

2. Understand the set of out-of-box association types that are shipped with Enterprise Manager and ensure the use of the most appropriate type.

   For more information, see About Out-of-Box Association Types.

3. Ensure that association derivation rules are used to (declaratively) describe the associations that are to exist based on configuration data that resides in the repository.

   Rules are triggered by configuration collections (where target property changes are also treated as a configuration collection).

   For more information, see Using Association Derivation Rules Syntax and Semantics.

4. You need to coordinate with the owners of other ME's regarding association maintenance, as associations with your ME types often involve other plug-in's ME types.

   Decide which plug-in will package the rules. The plug-in that owns the rule must ensure that it specifies all other needed plug-ins as prerequisites to ensure that all target types and their ECM metadata is present prior to rule installation.

   Advanced activation expressions, as described in Understanding Activation Expressions can be used if it is not possible to assure all needed target types are present. Rule triggers should reside in plug-ins that define target types specified by the triggers. However, if target types are known to exist before the plug-in installation, the triggers can reside along with the rule.

## Assumptions and Prerequisites

This chapter assumes you are familiar with the following:

- Association types in general, association type hierarchy, concepts of allowed pairs of manageable entity types for association types, forward and concrete (versus abstract) association type, and the semantics of the Enterprise Manager out-of-box association types.

- Target model, target properties, and target components.

- Enterprise Configuration Management configuration collections, including treatment of target properties as configuration data.

- Plug-in development overview, including how to package a plug-in and its XML files.

- Topology viewer (to view your associations).

# Understanding Enterprise Manager Association Concepts

In Enterprise Manager, the concept of a relationship is internally referred to as an association. An association (association instance) represents a relationship between two managed entities and specifies three values, namely, source, destination, and association type. For instance, in "`database1 exposed_by listener1`", `database1` is the source, `listener1` is the destination, and "`exposed_by`" is the association type.

This section describes association derivation rules, which provide a concise declarative means of defining association types. Association derivation (so called because the existence of associations is derived from collected data) provides a mechanism by which developers can cause association instances to be created and removed based on data collected from a target.

The association derivations are based on the data collected using configuration collections and present in the Enterprise Manager repository. The association derivation mechanism allows you to keep the association consistent with the collected configuration data and to determine associations centrally based on all known data (instead of being done by agent logic, which has access to less data).

## About Out-of-Box Association Types

Enterprise Manager provides a common set of association types that should meet the needs of most plug-in developers and you are encouraged to become familiar with these association types and use them if applicable.

The cardinality specifies the cardinality for the overall association type. An allowed_pairs (constraint) should not specify conflict cardinality, but may specify more specific cardinality. An abstract association type can not have association instance created for it.

The following diagram shows the core association type hierarchy. For more information on out-of-box association types, see Out-of-Box Associations

**Figure 11-1    Core Association Type Hierarchy**

## Using Association Derivation

To use association derivation, complete the following:

1. Specify the logic to run after the collection of target configuration.

   The logic derives a set of association instances in the form of triples that specify the source managed entity GUID, association type, and destination managed entity GUID. For instance, the association derivation logic for targets of type oracle_listener could return triples that represent associations between the listener and each database for which it listens.

2. Create and run a SELECT statement that contains the logic used to derive the triples.

   Each returned row contains association type, source, and destination columns and represents an association that should exist.

3. Register the derivation logic against an Enterprise Configuration Management snapshot type.

   After every snapshot collection, the registered logic is invoked. Input to the logic is the GUID of the target for which the data was collected.

When the association derivation logic for snapshot S of target T is executed, the derived associations replace the previously derived associations for snapshot S of target T. For example, if associations A1 and A2 were collected yesterday and only A1 is collected today, then A2 is effectively deleted.

## About Automated Discovery and Promotion of Associations

One option for adding associations to Enterprise Manager is to provide a discovery script which discovers targets and the associations between them, and the discovery script is then scheduled to run on a selected set of agents by the end-user. The targets and associations discovered by this type of script are automatically promoted, that is they are automatically added to Enterprise Manager. This approach is useful for associations that are between targets that are managed by your plug-in and therefore the specific target identification is known (that is you create the targets on both ends of the association). If these associations are to other targets not included in your plug-in, then typically a derived association rule is used to specify how to locate the "external" target.

A guided discovery process may be used if some interaction with the end-user or administrator is necessary to filter the information discovered by the script, or if some amount of post-processing is necessary to compare it to other information already known to Enterprise Manager.

## Understanding Association Creation During Guided Discovery

This approach is similar to the automated discovery approach described in the previous section in that you provide a discovery script that can be run by an Enterprise Manager agent. That discovery script may return any number of related targets and the associations between them. The difference is that in the guided discovery case, you provide a user interface that the end-user interacts with to drive the execution of the discovery script and then process the results returned from it. This processing

takes the output from the discovery script and may further filter it or present it to the end-user to allow them to add important information to it.

Guided discovery may also interact with the Enterprise Manager system using target services to obtain information about targets already known to Enterprise Manager to perform incremental updates to the topology of targets discovered. This approach is also used for cases where the associations to be created are between targets that are managed by your plug-in and therefore the specific target identification is known. That is you create the targets on both ends of the association, but some additional intervention is needed before those associations are added to Enterprise Manager.

# Using Associations Derived from a System Stencil

This approach is used solely for creating system membership associations between a system target and its members. The system target and its members are typically all part of a single plug-in, as you must have knowledge of the types of associations that exist between the system target and its members in order to form the system topology.

The system stencil defines the set of association paths that should be considered when forming the system membership. In this way, the plug-in can traverse complex association paths to locate targets that should be treated as members of the system. This is important in cases where a system member is not directly associated with the system target by some other "native" association.

If the plug-in does not include a target type that you wish to be treated as a system, then this approach can be ignored.

# Associations Derived from Rule

This approach for creating associations is particularly suited to cases where the destination target of the association is not part of the plug-in but is known to be managed by Enterprise Manager. For example, assume that the configuration of your target included a connection to an Oracle database that was used to store information related to your target operation (such as an application store). The configuration of your target knows something about the database that it uses, likely some connection related details such as host-port-sid or host-service.

You would like to represent this association between your target and the database in Enterprise Manager so that if Enterprise Manager is managing the database, the end-user can see this relationship and traverse it to obtain other information about that database and manage it (if appropriate and allowed).

Because you do not know if Enterprise Manager is managing the database and the identifying information you have is not the Enterprise Manager database target name, but instead the connect information, you can construct a derivation rule that maps the connection information in your target's configuration to that of a database in Enterprise Manager.

This approach is very useful for cases where you wish to construct this type of association between a target that is part of your plug-in and some external target, particularly some Enterprise Manager stack component like Oracle Fusion Middleware or the Oracle Database.

# About Association Derivation Rules Management

Since Enterprise Manager Cloud Control 12*c*, Enterprise Manager Cloud Control extends the use of associations by Enterprise Manager components and enhances the overall association framework. It introduces new consumers of associations, including the topology viewer.

Association framework enhancements include the treatment of associations as configuration data. Enterprise Configuration Management features such as change tracking and saved snapshots now apply to associations as well as to traditional configuration data. Associations can now specify source and destination target components, as well as target GUIDs.

The following sections provide detailed instructions on the use and management of derivation rules:

- Using Association Derivation Rules Syntax and Semantics
- Understanding XML Metadata File Syntax and Semantics
- Using Rule Semantics
- Maintaining Performance
- About Regular Query and Trigger Patterns
- Diagnosing Issues
- Useful Examples
- Applying the Mechanical Steps of Integration
- Understanding Activation Expressions
- Troubleshooting and Debugging

## Using Association Derivation Rules Syntax and Semantics

The following sections describe the contents of a rule, including name, query, triggers, and database objects that can be referenced by rule queries.

**Name**

A rule is identified by a unique rule name that must be unique across all plug-ins. Oracle recommends that you use a suitable prefix to avoid name conflicts. For example, a company symbol or name followed by the plug-in name.

**Query**

The primary component of a rule is the rule query, which identifies a set of associations. Each row returned by the query represents an association. The SQL must return four columns whose names and types must be:

- `assoc_type`

  (VARCHAR2(64)): the association type

- `source_me_guid`

  (RAW(16)): a managed entity GUID

- `dest_me_guid`

(RAW(16)): a managed entity GUID

- `derivation_target_guid, derivation_target_guid2, derivation_target_guid3`

  (RAW(16))

  Often unnecessary, these are one or more optional target GUID columns that identify targets involved in deriving the association (other than the source or destination).

  – These cannot be a target component ID, but must be a target GUID.

  – Columns should be used in order.

    This means that queries returning `derivation_target_guid2` must also return `derivation_target_guid`. Queries returning `derivation_target_guid3` must also return `derivation_target_guid` and `derivation_target_guid2` columns.

  The need in some cases for a derivation target guid is illustrated by the case in which the collection for a target determines associations between two other targets. For instance, the collection for a Siebel Enterprise System determines associations between its member targets.

  In this case, the derivation target GUID is the target GUID of the Siebel Enterprise System target, but the source and destination are other targets. Similar cases exist for Oracle E-Business Suite and Oracle WebLogic Server, where configuration information is collected from a single source, such as the Oracle WebLogic Server domain admin server, and used to derive associations between the domain members.

Each row returned by the query must specify a valid association instance and must use a concrete (not abstract) forward (not inverse) association type. Valid association instances must specify managed entities that are valid for the specified association type. For example, a `hosted_by` association must specify a destination that is a host target. Inverse association types must not be returned. For example, do not use `host_for`, which is the inverse of `hosted_by` and would be logged as an error.

Note that the rule query returns a repository-wide set of associations, but associations are populated incrementally on behalf of one target at a time. When the rule is evaluated, it is from the perspective of a single target. At evaluation time, the framework wraps the query with an outer query. For example:

```
"SELECT … FROM <query> WHERE derivation_target_guid = <initiatingTarget> AND …"
```

> **Note:**
>
> You do not need to specify a DISTINCT keyword (at the outmost level) in your rule query as the framework will eliminate the duplicates by itself when it wraps your query with its own query.
>
> Query size should not exceed 2000 characters. (It is planned to extend this to 4000 characters in a future release).

**DB Objects Referenced by Rule Queries**

For security reasons, the `SYSMAN_RO` user will execute your rule query. Therefore, only objects accessible by this user are allowed to be referenced. For objects

created outside of your plug-in you can reference views exposed by the Extensibility Development Kit (EDK) at your plug-in level, including those prefixed with `MGMT$`.

For objects created in your plug-in, you can reference `CM$` views auto-generated by the Enterprise Configuration Management framework for your target type collections. You can also reference views prefixed with a `DA_` prefix and packages with invokers rights with a `DA_` prefix.

Your query should not rely on associations unless you ensure that they are present by the time the query is executed (for example, when corresponding triggers fire – see below). For derived associations, the order of executions is not deterministic because the order in which configurations arrive and then corresponding associations are derived is arbitrary. An example of an association that *can* be used is a hosted_by association, which appears during target discovery and is not a derived association.

**Triggers**

Triggers are usually provided in addition to the rule query. A trigger specifies a table that, when changed, may impact the associations returned by the rule query. Generally there are multiple triggers because the rule query often refers to data from multiple tables and because changes to either target's data can affect the association rows returned by the query.

Two triggers may not always be needed as it may be the case that the data for one of the two targets does not change. For instance, an association rule that determines its destination based on (immutable) identity properties of one of the targets is only affected by changes to the source target's configuration. Even in that case, it may still be desirable to specify two triggers. If the destination target can appear after the source and this appearance causes the immediate creation of a new association, the trigger is needed.

A trigger specifies the following:

- A snapshot table

  A change to the table (due to upload of new data) will fire the trigger. You should only include tables that affect the set of associations because needless firing of triggers impacts performance. A table is identified by target type, snapshot type, and table name.

- Column ID flag

  This indicates whether the source, destination, or a derivation target guid should be used to identify associations affected by the newly uploaded configuration data. In other words, depending on this column value, associations for the source, destination, or a derivation target will be replaced with a new set of associations computed for that target when the trigger table data changes. Possible values include `source`, `destination`, `derivationTarget`, `derivationTarget2`, or `derivationTarget3`.

When the trigger fires, the association derivation framework will effectively replace all currently existing associations where the given target is a source, destination, or derivation (depending on the flag) with newly computed associations. To compute the new set of associations, the rule query is executed with the corresponding column bound to the target id. This simplified explanation assumes that associations only exist because of this single rule and it would be slightly changed for a target components case.

For example, a rule query accesses data from a listener configuration table and a database configuration table and returns associations of the form *<database>* exposed by *<listener>*. One trigger specifies the database configuration table and a column id flag of source, because a change to the configuration table for a database may affect rows where the database is the source of the association. Similarly, a second trigger specifies the listener configuration table and a column id flag of the destination, because a change to the configuration table for a listener may affect rows where the listener is the destination of the association.

If multiple rules can be triggered for a snapshot table, then the order in which the triggers execute is non-deterministic. This means that the developer cannot make any assumptions about the order.

The table name (TN) specified in a trigger can actually be the name of a base table, view, or synonym. In all cases, the underlying tables of TN are identified. A trigger is created for each such table that is an Enterprise Configuration Management table.

## Understanding XML Metadata File Syntax and Semantics

To create or update a rule, you edit an XML metadata file that defines the rule (or set of rules) and then import it into the repository. The metadata import is done when a repository is created or upgraded. It is also done when a plug-in is added, upgraded, or removed.

Schematically, you specify the following information for each rule in the file:

- Name
- Query
- 0 to *n* triggers with
    - Fully qualified snapshot type (which includes target type).
    - Metric table of that snapshot (view or synonym that refers to such a table)

      Normally, this view is used by the rule query.
    - Column flag (source, destination, derivationTarget, derivationTarget2, or derivationTarget3).
    - Optional details (used to specify target property names for triggers based on target properties).

The XML semantics are designed to describe the latest state of a rule and its triggers, no matter what the prior state was. So you only need the latest XML specification of a given rule to know how the rule and its triggers are specified for a plug-in. Moreover, one plugin will not be able to directly affect triggers of another plug-in. However, if a rule is removed, triggers referencing the rule from the other plug-ins will not be useful.

The following outlines the rule query specification semantics:

- A non-empty query implies that you need to add or, if needed, overwrite a prior rule query for a rule that had been registered by the same plug-in.
- Specifying no query implies removing the rule, if the rule query had been registered by the same plugin, and no change otherwise.

**Rule Location**

Once a rule R with its query is located in a file F within a plug-in P, its corresponding XML Rule element can never be removed from that file or from that plug-in (although

its attributes and subelements can be modified). Rule R will always be owned by plug-in P. If the rule does need to be removed, a Rule element with no query sub-element must remain in file F indicating that rule R had been removed. If it is important to move rule R to another file or plug-in, you must rename the rule (to R2 for example), remove rule R using the above syntax in file F, and add R2 in the new location. This will effectively remove R and create a new rule R2.

Plug-in P that owns a rule R should be chosen carefully to ensure that all target types needed by the rule query are present by the time plug-in P is installed. Rule R should rely only on targets of types that are always present in Enterprise Manager (for example, host), targets of types defined by plug-in P, or targets of types defined in plug-ins that are prerequisites of plug-in P. If it is impossible to chose such a plug-in, consider using the advanced feature of activation expressions discussed later.

### Trigger Specification Semantics

In terms of trigger specification semantics, you should replace triggers in the same plug-in with a new specified set of triggers. Alternatively, you can just remove any pre-existing triggers if the newly specified set is empty.

### Trigger Location

Normally triggers are defined as part of the rule definition in the same plug-in. This way, when the rule changes, corresponding triggers can also change if needed. However, in some cases it is preferable or only possible to place triggers in plug-ins defining the target types of the triggers. For example, a rule that computes association between targets and their corresponding Oracle Home targets cannot list all possible target types and corresponding triggers.

Instead, plug-ins owning target types that want to use the rule specify the triggers for the relevant target types. Therefore, a plug-in that owns an oracle_ias target type will have a trigger for this rule with oracle_ias listed as the target type in the trigger. Such triggers are changed or removed along with the corresponding plugin.

Once the rule's XML is listed in some file F in the plug-in that owns oracle_ias, it cannot move to another file (even if it specifies only triggers for the rule). Note that in our example, the rule query itself is not target type specific, as it only depends on the Oracle Home target type and not other specific target types. Therefore, the rule is defined in the plug-in P that is installed prior to the plug-ins (such as oracle_ias plugin). This way, when the trigger is imported into Enterprise Manager, it finds the rule already present.

The following points should also be taken into consideration:

- You should always have a rule and its triggers specified in at most one file for a given plug-in.

  For example, if some of the triggers for a rule (defined in a different plug-in P1) are specified in two files for plug-in P2, an import of the second file would overwrite the triggers that were specified in the first file. The order of import of the two files is not guaranteed.

- An error will result if a query is specified for a rule that has been registered by a different plug-in.

  In other words, plug-ins that have not specified a rule query can only specify a new set of triggers in their context, but cannot overwrite the query. Only one plug-in effectively owns the rule query.

- An error will result if there is a trigger specification for a rule that does not exist or is being removed (by not specifying the rule query).

- To effectively disable all triggers in all plug-ins for a given rule, the plug-in author can just remove the rule, using the syntax mentioned previously. If needed, the author can also create a rule with a different name as a replacement.

- To replace triggers, but not the rule query, in the plug-in that had specified the rule query in a prior release, specify the same query again and a set of new triggers in the next plug-in version.

> **Note:**
>
> In terms of performance, specifying textually the same query will result in the best upgrade performance, since the framework will not need to recompute all associations for the query.

The syntax for rule definitions is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="YesNo">
     <xs:annotation><xs:documentation>
         Type definition for the Yes/No atribute value.
     </xs:documentation></xs:annotation>
     <xs:restriction base="xs:NMTOKEN">
         <xs:enumeration value="Y"/>
         <xs:enumeration value="N"/>
     </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="NameDef">
     <xs:restriction base="xs:string">
        <xs:pattern value="[A-Za-z][A-Za-z0-9_]*"/>
     </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="TriggerKind">
     <xs:restriction base="xs:string">
        <xs:enumeration value="C"/>
        <xs:enumeration value="H"/>
     </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ColumnID">
     <xs:restriction base="xs:string">
        <xs:enumeration value="source"/>
        <xs:enumeration value="destination"/>
        <xs:enumeration value="derivationTarget"/>
        <xs:enumeration value="derivationTarget2"/>
        <xs:enumeration value="derivationTarget3"/>
     </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="RuleContentWFlags">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="source_comp" type="YesNo" use="optional">
          <xs:annotation> <xs:documentation>
            Can source entity be a target component? Default: No
```

```
          </xs:documentation> </xs:annotation>
        </xs:attribute>
        <xs:attribute name="dest_comp" type="YesNo" use="optional">
          <xs:annotation> <xs:documentation>
            Can destination entity be a target component? Default: No
          </xs:documentation> </xs:annotation>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="RuleType">
    <xs:annotation> <xs:documentation>
      Rule definition.
    </xs:documentation> </xs:annotation>
    <xs:sequence minOccurs="0">
      <xs:choice>
        <xs:element name="query" type="RuleContentWFlags" minOccurs="0"
          maxOccurs="1">
          <xs:annotation> <xs:documentation>
            Query that returns 1 row  per association.  Must return
            columns named ASSOC_TYPE, SOURCE_ME_GUID, DEST_ME_GUID, and
            optionally, one or more of DERIVATION_TARGET_GUID,
            DERIVATION_TARGET_GUID2, DERIVATION_TARGET_GUID3.
            Returning DERIVATION_TARGET_GUID[N] column
            implies the query also returns DERIVATION_TARGET_GUID and all
            DERIVATION_TARGET_GUID[K] for all K between 2 and N.
          </xs:documentation> </xs:annotation>
        </xs:element>
      </xs:choice>
      <xs:element name="trigger" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="targetType" type="xs:string" minOccurs="1"
             maxOccurs="1"/>
            <xs:element name="snapshotType" type="xs:string" minOccurs="0"
             maxOccurs="1"/>
            <xs:element name="table" type="xs:string" minOccurs="0"
              maxOccurs="1">
              <xs:annotation> <xs:documentation>
                Name of an ECM table or more likely a view whose query
                relies on ECM snapshot table(s). The table(s), when
                uploaded,
                should trigger evaluation of the rule.  (The fully
                qualified name includes target type and snapshot
                type.)
              </xs:documentation> </xs:annotation>
            </xs:element>
            <xs:element name="idColumn" type="ColumnID" minOccurs="1"
              maxOccurs="1">
              <xs:annotation> <xs:documentation>
                Indicates whether source, destination, or a derivation
                target
                should be used to identify associations affected by the
                newly
                uploaded configuration data. In other words, depending on
                this
                column value, associations for the source, destination,
or
                a derivation target will be replaced with new set of
                associations computed for that target, when the trigger
                table data changes.
```

```
                                          ColumnID type definition contains allowed values.
                                      </xs:documentation> </xs:annotation>
                                  </xs:element>
                                  <xs:element name="details" type="xs:string" minOccurs="0"
                                     maxOccurs="1">
                                     <xs:annotation> <xs:documentation>
                                        Additional details for the trigger. Currently used for
                                        target properties table, in which case, it contains comma
                                        separated list of property names that should fire the
                                        trigger. Absence
                                        of property names indicates that any property change
would
                                        fire the trigger (for the given target type).
                                        Note: white space is ignored.
                                     </xs:documentation> </xs:annotation>
                                  </xs:element>
                              </xs:sequence>
                              <xs:attribute name="kind" type="TriggerKind" use="optional">
                                  <xs:annotation> <xs:documentation>
                                     Kind of the trigger. "C" (configuration load trigger) by
                                     default.
                                     Other allowed value: "H" (host change trigger)
                                  </xs:documentation> </xs:annotation>
                              </xs:attribute>
                          </xs:complexType>
                      </xs:element>
                  </xs:sequence>
                  <xs:attribute name="name" type="NameDef" use="required">
                      <xs:annotation> <xs:documentation>
                          Name of rule, which must be unique.  Recommendation: Use a
                          prefix that identifies your plugin.
                      </xs:documentation> </xs:annotation>
                  </xs:attribute>
                  <xs:attribute name="activation_expr" type="xs:string" use="optional">
                      <xs:annotation> <xs:documentation>
                          Optional activation expression. If not present or empty, implies
                          that the rule is always active. Else, the value is a Boolean
                          activation expression which must produce true if and only if
                          the rule should be active.

                          The expression can use (case insensitive) "AND", "OR", and
                          parenthesis. Operands of the expression are target types.
                          Each occurrence of a target type evaluates to "true" if
                          and only if the target type is present in EM.

                          Note that a number of target types do not need to be listed in
                          the expression because they are always going to be present
                          whenever the rule is installed and present in EM installation.
                          These include:
                          - target types installed with the plugin where the rule resides
                          (i.e. target types in the plugin which owns the rule)
                          - target types in other plugins on which the plugin owning
                          the rule depends
                          - target types always installed with EM (like host)
                          Thus, in many cases, if this option is used, a single target
                          type, as in example 1 below, may suffice.

                          Examples:

                          (1) "oracle_ovm"
                          This simple expression implies that the rule should be
```

```
                   active only if oracle_ovm target type is installed at EM
                   (in addition to any other target types that are already known
                   to be present when this rule is installed).

                   (2) "oracle_ovm and (oracle_oam_cluster or oracle_oim_cluster)"
                   This expression implies that the rule should be active only if
                   oracle_ovm target type is present and either oracle_oam_cluster or
                   oracle_oim_cluster is also present.
               </xs:documentation> </xs:annotation>
           </xs:attribute>
       </xs:complexType>
       <xs:element name="Rules">
           <xs:complexType>
               <xs:sequence minOccurs="1" maxOccurs="unbounded">
                   <xs:element name="Rule" type="RuleType"/>
               </xs:sequence>
           </xs:complexType>
       </xs:element>
   </xs:schema>
```

**For example**:

```
<?xml version="1.0" encoding="UTF-8"?>
<Rules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Rule name="ora_listensFor">
    <query>
      SELECT ...
      fill in query
    </query>
    <trigger>
      <targetType>oracle_database</targetType>
      <snapshotType>db_config</snapshotType>
      <table>CM$DB_CONFIG_TABLE</table>
      <idColumn>destination</idColumn>
    </trigger>
  </Rule>
</Rules>
```

## Using Rule Semantics

The following algorithm depicts a simplified form of the semantics for rule R, where the trigger specifies a query $Q$ and a flag $FC$ that corresponds to a column name of either source, destination, or derivation target GUID.

When updates to a snapshot table are uploaded for some target $t$ with a GUID of $t\_guid$, for each trigger that specifies the modified table, execute these statements:

```
DELETE FROM MGMT_ASSOC_INSTANCES
 WHERE <FC> = t_guid AND RULE_ID = R

INSERT INTO MGMT_ASSOC_INSTANCES
(SELECT a.*, R
  FROM (<Q>) a
 WHERE <FC> = t_guid )
```

The actual implementation differs from the above example for the following reasons:

- The actual implementation will not delete and then add the same association as this would be inefficient.

Rather it will compare the current and new set, making changes only where needed. Moreover, an actual delete statement also deletes any associations that specify a target component of the target.

- It is possible that an association may be asserted by more than one origin

  This is managed using an origins table, whose contents are rolled up into the `MGMT_ASSOC_INSTANCES` table.

- Validity testing is performed at various points.

  For example, to test that the association type is valid for the provided source and destination MEs.

## Maintaining Performance

Because the evaluation of derivation rules may be frequent, any poor performance of the rule queries can be problematic. Rule authors must ensure that any needed indexes are present and that they test query performance based on the specific queries that are generated for each trigger.

In particular, testing of the rule query must be done for each trigger because each trigger causes the execution of a different query. Note how rule query return values are bound to a given target globally unique identifier (GUID) depending on your triggers.

You must have indexes that will make use of these bindings. Furthermore, queries should be written in such a way that they would not prevent the push of bindings from outside into your queries.

## About Regular Query and Trigger Patterns

The following sections outline the regular patterns you would normally see in queries and triggers. You should check whether your queries and triggers adhere to these patterns and if not, document the reasons why (since such cases normally represent the exceptions from the rules of thumb).

**Query Patterns**

The following outline common query patterns:

1. The derivation target should be non-null when the ECM configuration of such target is used to derive associations between two other entities.

   One known case for the use of derivation targets is associations with a database system. The database instance target provides configuration, while associations are made with the corresponding database system target. In such cases, the database instance must act as a derivation target for the associations to the corresponding database system target.

2. The query must only access objects such as `CM$`, other views that access Enterprise Configuration Management data, or views that access target information, such as `MGMT_TARGETS`.

   For more information on the objects that can be accessed, see Using Association Derivation Rules Syntax and Semantics.

3. Association types must be forward and concrete.

**Trigger patterns**

The following outline common trigger patterns:

1. The number of triggers will often be equal to the number of non-target-entity views in the FROM clause. In other words if the `mgmt_targets`, `mgmt_target_properties`, `mgmt$target`, `mgmt$target_properties`, and other such views are disregarded.

   Each Enterprise Configuration Management view will correspond to one trigger. One exception is when a table may be triggered from more than one target type (for example, `oracle_database` and `rac_database`). In this case, multiple triggers for the same Enterprise Configuration Management view could be supplied.

2. The table name of the trigger must be based on Enterprise Configuration Management metadata tables for the snapshot specified by the trigger's target type and snapshot type.

   Normally, it should be one of the objects in the rule query FROM clause (for example, a `CM$` view).

3. A view specified in the trigger is joined (perhaps indirectly) in the query with a target (or target component) entity.

   The entity id will be returned as source, destination, or derivation target in the select clause. The idColumn will match this.

   For example, association between targets A and B is dependent on a join between the `cm$Aconfig` and `cm$Bconfig` tables, where data from the `cm$Aconfig` table comes from target A and data from the `cm$Bconfig` table comes from target B. The trigger for the `cm$Aconfig` table will have an idColumn matching target A (for example, source) and the trigger for the `cm$Bconfig` table will have an idColumn that matches where target B GUID is returned (for example, destination).

4. The trigger target type must match the target type of the target returned by the query in the column specified by idColumn.

   More generally, the target type of the target returned by the rule query could be a subtype or a target component of the trigger target type.

5. If trigger relies on target properties, specific property names should be identified in the details tag.

## Diagnosing Issues

To help diagnose issues and understand how associations were derived, the framework records information about how associations were derived when in debug mode. For more information on debug, see Troubleshooting and Debugging . It also includes additional sanity (error) checking. For instance, one test checks that the derivation target GUID is that of a real and current target.

## Useful Examples

While the following examples include the use of target properties to illustrate their proper employment, Oracle does not recommend relying on target properties. Instead, configuration data should be properly modeled using ECM tables. For more information, see Are there guidelines for when to use target properties? .

When reviewing these examples, it is helpful to remember the following concepts:

- Every target type has an Enterprise Configuration Management snapshot type called `orcl_tp_config`.

It includes a snapshot table referenced by the `GC$TARGET_PROPERTIES` view, which if needed should be used by the triggers. Current data for target properties can be accessed through the `MGMT_TARGET_PROPERTIES` and `MGMT$TARGET_PROPERTIES` EDK objects.

- Every Enterprise Configuration Management snapshot table will by default have a view for accessing the current configuration data.

  Its name will be that of the table, with the prefix `CM$`. Most rule queries will refer to configuration tables through their `CM$` views.

## Host on a Virtual Machine

A 'deployed_on' association type is used to represent the fact that a host target is deployed on a virtual machine target.

The Query below returns all associations between a host and associated virtual machines based on matching their MAC addresses. Triggers are defined so that they trigger the rule whenever the corresponding configuration view (that includes the MAC address) changes. The rule described below would reside in the plug-in defining virtual machine (host target type is guaranteed to be present on any EM installation). Both triggers can be included in the rule and would belong to the plug-in defining virtual machine.

```
<Rule name="...">
    <query>
        select 'deployed_on' as assoc_type,
                host.target_guid as source_me_guid,
                guest.cm_target_guid as dest_me_guid
          from mgmt$hw_nic host,
                cm$vt_vm_vnic guest
         where guest.mac_address = host.mac_address_std
    </query>
    <trigger>
        <targetType>host</targetType>
        <snapshotType>ll_host_config</snapshotType>
        <table>MGMT$HW_NIC</table>
        <idColumn>source</idColumn>
    </trigger>
    <trigger>
        <targetType>oracle_vm_guest</targetType>
        <snapshotType>ovm_guest_config</snapshotType>
        <table>CM$VT_VM_VNIC</table>
        <idColumn>destination</idColumn>
    </trigger>
 </Rule>
```

## Target installed_at Oracle Home

The Oracle Home target type includes the `INSTALL_LOCATION` target property that contains the name of the directory in which the Oracle Home resides. For all target types that are installed in Oracle homes, there is an `OracleHome` target property that specifies the same value as `INSTALL_LOCATION`. Whenever a target's `OracleHome` value matches the `INSTALL_LOCATION` value and both targets reside on the same host, an `installed_at` association exists.

Both a target's `OracleHome` and a home's `INSTALL_LOCATION` are subject to change. It is also possible for a target or home to be created that in turn matches up with a home or target. However, the value of a target's host is immutable.

**Query**

Returns all associations between Oracle Home targets and the targets that are installed in them.

```
<Rule name="...">
   <query>
        select 'installed_at' as assoc_type,
        t.target_guid as source_me_guid,
        o.target_guid as dest_me_guid
    from mgmt_targets t,
        mgmt_targets o,
        mgmt_target_properties tp,
        mgmt_target_properties op
    where o.target_type = 'oracle_home' and
        t.host_name = o.host_name and
        tp.target_guid = t.target_guid and
        tp.property_name = 'OracleHome' and
        op.target_guid = o.target_guid and
        op.property_name = 'INSTALL_LOCATION' and
        tp.property_value = op.property_value
   </query>
    <trigger>
        <targetType>oracle_home</targetType>
        <snapshotType>orcl_tp_config</snapshotType>
        <table>GC$TARGET_PROPERTIES</table>
        <idColumn>destination</idColumn>
        <details>INSTALL_LOCATION</details>
    </trigger>
</Rule>
```

**Trigger 2**

The following trigger for the same rule would reside in the plug-in that defines oracle_database target type:

```
<Rule name="...">
    <trigger>
       <targetType>oracle_database</targetType>
       <snapshotType>orcl_tp_config</snapshotType>
       <table>GC$TARGET_PROPERTIES</table>
       <idColumn>source</idColumn>
       <details>OracleHome</details>
    </trigger>
</Rule>
```

Trigger 3-*n*

This is the same as trigger 2 only with another target type that has an `OracleHome` property. These triggers would reside with plug-ins that define corresponding target types. This trigger has the same characteristics as trigger 2, except it uses a different target type that has an `Oracle_Home` property.

## Listener and Database

An `exposed_by` association type is used to represent the fact that a database is exposed by a listener to applications. One way that this association can be created is based on the ports for which the listener is configured.

**Query**

Returns all associations between a database and listener on the same machine such that the ports match. Both triggers can reside with the rule in the plug-in that defines Oracle database and Oracle listener target types.

```
<Rule name="...">
    <query>
        select 'exposed_by' AS assoc_type,
               oradb.target_guid AS source_me_guid,
               oralsnr_ports.cm_target_guid AS dest_me_guid
          from mgmt_targets oradb,
               mgmt_target_properties oradbprops1,
               mgmt_target_properties oradbprops2,
               cm$listener_ports oralsnr_ports
         where oradb.target_type = 'oracle_database'
           and oradb.target_guid = oradbprops1.target_guid
           and oradbprops1.property_name = 'MachineName'
           and oradbprops1.property_value = oralsnr_ports.machine_name
           and oradbprops1.target_guid = oradbprops2.target_guid
           and oradbprops2.property_name = 'Port'
           and oradbprops2.property_value = oralsnr_ports.listener_port
    </query>
    <trigger>
        <targetType>oracle_database</targetType>
        <snapshotType>orcl_tp_config</snapshotType>
        <table>GC$TARGET_PROPERTIES</table>
        <idColumn>source</idColumn>
        <details>MachineName</details>
    </trigger>
    <trigger>
        <targetType>oracle_listener</targetType>
        <snapshotType>listener_config</snapshotType>
        <table>CM$LISTENER_PORTS</table>
        <idColumn>destination</idColumn>
    </trigger>
</Rule>
```

# Applying the Mechanical Steps of Integration

After you decide on the ME/association model and write the rules, proceed with the implementation as follows:

1. If your rules need Enterprise Configuration Management configuration data not yet present, add new Enterprise Configuration Management metrics or extend the existing ones.

   If needed, you should add new Enterprise Configuration Management tables or columns. You must also make sure that the default collection schedule specifies *<Schedule OFFSET_TYPE="INCREMENTAL">*. Failure to do so will delay the loading of the configuration such that, for instance, newly discovered targets may not get associations for thirty minutes or more.

2. The Association types framework has the concept of allowed pairs indicating which target types are allowed to be associated by a given association type. If you are creating associations between ME types that are not listed as allowed pairs for the respective association type, add the needed pairs.

3. Create one or more files to define the association derivation rules. Syntax errors, such as failing to conform to the XSD, are passed through as Java exceptions. You may want to use JDeveloper or another tool to confirm that you have created a valid document.

4. Test the rule files by importing them using the following command:

```
emctl register oms metadata -sysman_pwd sysman -pluginId <your.plugin.id>
-service derivedAssocs -file <fileName>
```

Validity testing is performed, so diagnostics may result.

5. Package the files into your plug-in.

Place them so that they are imported at repository creation or upgrade time in accordance with the conventions defined by the metadata framework. If part of a plug-in, place the files in a location similar to:

```
<stage_dir/plugin_dist>/oms/metadata/derivedAssoc
```

6. Test the derivation rules with cases that exercise every rule trigger that you specified.

One option is to initiate the upload of the Enterprise Configuration Management configuration data and check that the associations are properly established. Alternatively, you can directly call the PL/SQL procedure that will trigger the rules:

```
DECLARE
    temp GC$DERIV_ASSOC_CHANGE_LIST := GC$DERIV_ASSOC_CHANGE_LIST();
BEGIN
    GC$ECM_CONFIG.run_assoc_deriv_rule(
      p_target_guid => hextoraw('CC70BC294B82E7E9A95DFC257CFA6459'),  --
      Updated target/ME guid
      p_rule_name => '...', -- your rule name
      p_column_flag => 'D', -- column flag specifying the
      perspective from which to fire the rule. Possible values:
      S|D|T|U|V (implying source,destination, derivation target,
      derivation target 2, or derivation target 3, respectively)
      p_change_list => temp);
      COMMIT;
      -- examine p_change_list if needed
END;
```

> **Note:**
>
> Test the performance of your queries for each trigger after the corresponding output of the query has been bound, as described in Ensuring Performance.
>
> Use the import utility to make rule changes and try again.

## Special Triggers: Host Name Change Triggers

With this release, a new kind of trigger is supported for when the host name of a target changes. If your query relies on the host_name column of the mgmt_targets table, this trigger can be useful as it will fire when the host_name column changes, for example, upon relocation of the target.

The trigger syntax specifies a "kind" attribute of the "trigger" element with a value of "H" (which stands for "Host change" trigger). Trigger sub-elements will specify target type, which the trigger applies, and idColumn, which identifies the perspective from which to evaluate the rule. Possible values for idColumn are the same as those for regular triggers.

For example:

```
<trigger kind="H">
  <targetType>oracle_database</targetType>
  <idColumn>source</idColumn>
</trigger>
```

This specifies that the rule (which the trigger is part of) has to be reevaluated from the source perspective whenever oracle_database target's host_name column changes.

In general, the same rules apply to host change triggers as to regular triggers, including applicable trigger patterns (such as trigger pattern 4, which indicates that the column returned by the query corresponding to idColumn should be type or subtype of the targetType element). The rules related to trigger lifecycle and regular trigger placements in files and plug-ins also apply to host name change triggers.

## Understanding Activation Expressions

As described previously, rules are normally owned by the plug-ins that require all target types needed by the rule query to be present by the time the rule is installed. However, on rare occasions you may encounter a case where two or more plug-ins needed by a rule query are independent and any one of the plug-ins may exist without the presence of the other. In other words, it may not be possible to specify that one plug-in is a prerequisite of another for a given rule query that relies on configuration tables and data from target types of both plug-ins.

For such cases, you can specify an activation expression in a rule that will indicate when the rule should be active. Note that the rule is still owned by (at most) a single plug-in and the rule query can only be specified in one plug-in that will in the future be responsible for changing or removing the rule. However, the rule could be inactive for as long as not all needed target types are present on the system.

In terms of syntax, you specify activation expression using an attribute in the Rule element where the rule's query is specified:

```
<Rule name="..." activation_expr="..."> ...
```

Normally, when the activation_expr attribute is not present, it implies that the rule should always be active. If it is present, its value is a Boolean expression which must produce true if and only if the rule should be active. The expression can use (case insensitive) "AND", "OR", and parenthesis. Operands of the expression are target

types. Each occurrence of a target type evaluates to "true" if and only if the target type is present in Enterprise Manager.

Note that a number of target types do not need to be listed in the expression because they are always going to be present whenever the rule is installed and present in Enterprise Manager installation. These include:

- target types installed with the plug-in where the rule resides (target types in the plug-in that owns the rule).

- target types in other plug-ins on which the plug-in owning the rule depends.

- target types always installed with Enterprise Manager (like host).

Therefore, in many cases when activation expression is used, a single target type as described in Example 1 below may suffice:

1. **Example 1: oracle_ovm**

    This simple expression implies that the rule should be active only if oracle_ovm target type is installed in Enterprise Manager (in addition to any other target types that are already known to be present when this rule is installed).

    This kind of activation expression could be expected in a rule with a query that relies on configuration tables of oracle_ovm and oracle_xyz (for example) target types. Assuming these target types belong to different and independent plug-ins, if the rule is placed in a plug-in owning oracle_xyz target type, its activation expression would be oracle_ovm.

2. **Example 2: oracle_ovm and (oracle_oam_cluster or oracle_oim_cluster)**

    This expression implies that the rule should be active only if the oracle_ovm target type is present and either oracle_oam_cluster or oracle_oim_cluster is also present.

Please note that activation expressions should be used very carefully and rarely, since their usages are error prone due to lack of checks prior to rule activation. For example, any typo in a target type or any logical expression error may result in the rule never being activated or not being activated in correct cases. Enterprise Manager cannot check for validity of target types because it will assume unknown target types in the expression may get installed in the future.

The following describes how the activation expression feature interacts with other derived association features:

- During a new release of a rule XML, if rule query is unchanged but the activation expression is changed, the activation expression is updated and the rule is activated or deactivated if needed. If the rule is activated or deactivated, the rule's association instances are reevaluated or removed, respectively.

- Whenever Enterprise Manager adds or removes a target type (due to installation or deinstallation of a plugin for example), Oracle will reevaluate relevant activation expressions and activate or deactivate corresponding rules accordingly.

    Note that target type addition is performed before any corresponding targets and their associations or data are added. Target type deletion is done after target instances and their associations are removed. Therefore, we do not reevaluate corresponding association instances for the affected rules. By the time the rule is activated due to the addition of a target type, no associations should exist for such a rule.

Similarly, when the rule is deactivated due to the removal of a target type, the associations are also removed because all targets of that target type are removed. This logic applies to all known cases, including when the target types in the expression are those of source, destination, or one of the derivation targets. Thus, there is no reevaluation of the rule upon target type addition or removal.

- Note that there is a difference between the quarantine feature and the activation expressions. The quarantine feature is controlled by the end-users or administrators to decide which rule evaluations to turn off. On the other hand, activation expressions are controlled by the rule authors and a given Enterprise Manager setup (for example, the presence or absence of involved target types).

  Rules that were never activated cannot be quarantined. Otherwise, all other combinations are supported. For example, if a quarantined rule is deactivated and then activated again using an activation expression, it stays quarantined.

  Similarly, if an active rule gets quarantined by an administrator and later becomes deactivated due to a target type removal, administrators can still unquaratine it so that if it ever gets activated, it will start computing associations.

- When a rule is being removed, rules activation expression and its activation status are ignored. In other words, a rule can be removed even if it is inactive.

In general, there are four kinds of version specifications that are supported:

- "target_type[version], e.g. "oracle_ovm[3.7]"

  Indicates that this part of the expression is true only if the specified version of the target type is present. In above example, if oracle_ovm target type version 3.7 is not installed, the expression will evaluate to false even if other versions of oracle_ovm are present.

- "target_type[version1-version2], e.g. "oracle_ovm[3.7-5.2]"

  Indicates that this part of the expression is true only if a version of the target type is present that is between the indicated versions, including the two specified versions. Thus, in above example, the expression would be true if oracle_ovm target type of version 3.7, 3.8, 4.5, 5.0, or 5.2 is present.

  If, on the other hand, none of the installed versions of oracle_ovm target type fall into the range between 3.7 and 5.2, "oracle_ovm[3.7-5.2]" would evaluate to false.

- "target_type[version-], e.g. "oracle_ovm[3.7-]"

  Indicates that this part of the expression is true only if a version of the target type is present that is the same as the specified version or greater than the specified version. This is the most commonly used variation and the specified version would normally be the one where a table used by the rule query is introduced into a target type collection. In above example, the expression will evaluate to true if and only if a version of 3.7 or higher for target type "oracle_ovm" is installed at Enterprise Manager.

- "target_type[-version], e.g. "oracle_ovm[-5.2]"

  Indicates that this part of the expression is true only if a version of the target type is present that is the same as the specified version or less than the specified version. Thus, in above example, the expression will evaluate to true if and only if a version of 5.2 or less for target type "oracle_ovm" is installed on Enterprise Manager.

> **Note:**
>
> There are no spaces between target type and the opening square bracket.
>
> Version specification is optional. You can use just target type specification implying that any version of the target type would satisfy that part of the expression.

For example:

```
"oracle_ovm[3.7-] and (oracle_oam_cluster or oracle_oim_cluster[-2.5]
or oracle_oim_cluster[2.8-])"
```

This expression implies that the rule should be active only if oracle_ovm target type of version 3.7 or higher is present, and either oracle_oam_cluster (of any version) or oracle_oim_cluster of versions 2.5 and below or 2.8 and above is also present.

## Troubleshooting and Debugging

You can begin debugging by initiating the configuration collections used to fire your triggers. These collections occur when a target is used for the first time or whenever you make changes to the configuration data contained in the tables or views specified in your triggers. You can restart the Management Agent to recollect the data. Make sure that the data in your configuration tables changes as expected before checking whether the triggers fired.

Check that your rule query produces all the required associations across your development Enterprise Manager repository.

Finally, you can manually run the `GC$ECM_CONFIG.RUN_ASSOC_DERIV_RULE` PL/SQL API to manually create the required associations as if the rule did fire during the configuration change.

If the associations you expect are not created, then:

1. Make sure your query produces the required association when run manually and binding a source or destination GUID to the correct target.
2. Check for errors in relevant error tables mentioned in this section.

If this does not help, then you must figure out where the process is failing. This can be any of the following:

- The configuration collection is not collected
- The configuration collection is not changing in the place specified by the trigger
- The trigger firing resulted in an error
- The trigger that did fire is not producing the required association
- The trigger firing action did not get processed yet because of a queue backlog

**Troubleshooting Tips**

The following list provides tips about how to investigate your issue with associations not being created:

- Make sure that the rule is active and not quarantined in your environment:

```
select r.rule_name, q.column_flag, r.is_active,
       case when q.quarantined_time is null
          then 'No' else 'Yes' end as is_quarantined
from mgmt_deriv_rules r, mgmt_deriv_rule_queries q
where r.rule_id = q.rule_id and r.rule_name = your_rule_name;
```

- Check for derived association-related errors in the Management Repository:

```
select *
  from mgmt_system_error_log
  where module_name = 'EM.deriv'
  order by occur_date desc
```

Optionally, you can add an "and error_msg like '%<your rule name>%'" condition or use other substrings related to your association to limit the results, if there are too many that seem unrelated.

For example, if you see a message containing "ORA-20624: Specified assoc does not match any constraint assoc type", this implies that the allowed type pair for this association type and source/destination target types was not registered in the repository.

- If you can reproduce the issue, turn on additional logging and call one of the following:

```
EMDW_LOG.SET_TRACE_LEVEL('EM.deriv', EMDW_LOG.LINFO); COMMIT;

EMDW_LOG.SET_TRACE_LEVEL('EM.deriv', EMDW_LOG.LDEBUG); COMMIT;
```

DEBUG is very verbose and includes the queries used.

> **Note:**
>
> – The module name is EM.deriv.
>
> – New sessions get a new level. Existing sessions, such as long-running sessions, are not affected by the change as implemented in EMDW_LOG.
>
> – You can turn logging off using the constant EMDW_LOG.LOFF.

- View the log.

Logging is performed on the EMDW_TRACE_DATA table. Use this query to view the log:

```
SELECT log_timestamp, TRIM(log_message)
FROM   emdw_trace_data
WHERE  module = 'EM.deriv'
ORDER BY log_timestamp ASC
```

> **Note:**
>
> Adding conditions such as "log_message like" and "log_message like '%<your rule name>%'" condition can reduce your results.

- Check the log to confirm that the correct rules are getting triggered.

  Look for the line "Resulting action list:" followed by a line for each action that specifies the rule and column flag S|D|T|U|V.

- Determine which trigger should have fired and for which target your required association should have been created. For example, check the saved_timestamp in the MGMT$ECM_CURRENT_SNAPSHOTS view for your snapshots that trigger one of your triggers to see which one changed and was last saved.

  Next, check if the trigger did not fire because of a queue backlog. This applies to larger sites that are more likely to have backlogs in execution queues.

  1. Check the derived association queue for retry actions:

     ```
     select * from em_deriv_retry_actions
      where is_pending = 'T'
        and rule_id = (select rule_id from mgmt_deriv_rules
                          where rule_name = <'<your rule name>')
        and target_guid in (list of hextoraw(<target_guid on both ends of your
                              missing associations - or just the target you
     found
                              should have triggered the evaluation>);
     ```

     If this statement returns rows, then evaluation of the trigger is still pending due to the backlog.

  2. If you have a trigger that relies on target properties, then check the target properties queue to see if the system has processed all the follow-up actions for a given target property update:

     ```
     select * from EM_TPROPS_PENDING

      where target_guid = hextoraw(<target guid of the target with changed
                                 target property>);
     ```

     This query returns all the yet-unprocessed target properties in the queue for which the trigger would not have fired yet.

- Test your rule query (if debugging is turned on as described in a previous bullet point).

  In the emdw_trace_data table, find the variant of the query executed immediately after the line that reads `After query:` Try to execute it, replacing `:x` with `HEXTORAW(target guid)`. The GUID can be found earlier in the log on the `vvvvvvvvvvv RUN_SNAPSHOT_RULES` line.

  It should return a row for each association that should exist based on the specified rule and the collection for your target. If not, check that you did not enter an incorrect query or specify the wrong flag on the trigger.

  You can try variants on the last line. For example:

  ```
  WHERE source_me_guid = :y
  WHERE dest_me_guid = :y
  WHERE derivation_target_guid[N] = :y
  ```

- After you have logged the rule query, the log reflects the rows in the `MGMT_ASSOC_INST_ORIGINS` table that need to be changed, followed by the actual association rows in `MGMT_ASSOC_INSTANCES`.

- Check that the association instances are present in `MGMT_ASSOC_INST_ORIGINS`.

  If so, the derived association rule listed in column `DERIVATION_RULE_ID` has correctly asserted the existence of that association. Something is preventing the association from being created. Is the association type correct, and are the allowed type pairs registered with the association framework? Were any exceptions thrown?

# Ensuring Performance

The rules you provide may be fired frequently, depending on the triggers you define and the change frequency for the corresponding configuration tables. Poor performance of frequently triggered rules can adversely affect overall OMS operation.

A rule query is mapped to more specific queries. The query that gets executed depends on the column flag of the trigger (source, destination, or derivationTarget[N] column). As noted previously, the rule query `<Q>` is mapped to a query of this form:

```
SELECT a.*,
  FROM (<Q>) a
 WHERE <FC> = ?
```

where the parameter is the GUID of the target for which the Enterprise Configuration Management collection fired the trigger and `<FC>` is the source/destination/derivationTarget[N] column of your query specified by the trigger.

As you can see, the overall query (<Q>) will be filtered based on one of the target GUIDs that it returns. This means that query plans will generally start with data for that target and perform joins from there. Your query plans must push the "<FC> = ?" predicate all the way down and start evaluation with this predicate. Normally, they contain many nested loops that on the deepest level perform above binding and then get to the rest of the data starting from there via indexed lookups. Normally, there should be no full table scans of potentially large data tables (or hash joins).

Consider the example in Listener and Database. When the first trigger fires, the query will bind a database target and look for associated listener targets. The only way the Enterprise Manager repository can find the other end of the association (for example, the listener targets) is via "cm$listener_ports oralsnr_ports" joins on the machine_name and listener_port columns. If the table under view cm$listener_ports can be large, rule author should ensure an index exists starting with these two columns to quickly locate the listener targets instead of performing a full table scan on the table.

For each trigger, you must ensure that supporting indexes are present and that you test the performance of your queries after surrounding them in the query, as described earlier. However, if for example you have multiple triggers for the source column flag, you may have to test the performance only once as the generated query will be the same for both queries.

## Using Custom Configuration Specifications for Data Collection

Skip this note if you are not familiar with Custom Configuration Specifications (CCS). CCS tables are generic so that they can accommodate a variety of data and so tend

not to perform well for querying. However, properly modeled ECM tables are specific to the data being collected and can perform well for critical code paths, like derived associations computation code. Therefore, you should not use CCS collected data for derived association rules. Instead, collect the data required for derived associations separately using standard ECM collections.

# Using Overlapping Associations

It is possible for more than one rule to derive the same association, although usually you should avoid creating such overlapping rules. This section describes what happens when an association is derived by multiple rules and includes suggestions on when to avoid this and how.

## Overlap Between Associations Derived by Rules

When more than one rule derives the same association, that association continues to exist until each rule no longer derives it.

Sometimes, this is what you want. For instance, suppose each of two application target types has knowledge of both the Oracle WebLogic Server on which it runs and the database it accesses. Based on that knowledge, each has a way to derive an association between the Oracle WebLogic Server and the database. If either rule derives the association, that association is real and should exist. Only when both rules no longer derive the association can you be sure that the association no longer exists.

The "exists when any rule derives it" semantics may not be what you intend. Consider two rules that could be defined for the `installed_on` association between the database and Oracle home. Both access the same data, but one is triggered by a property change to the Oracle home and the other by a change to the database. As soon as either rule determines the relationship is gone, the association should be deleted. In such a case, you should use a single rule with two triggers.

Suppose you did not take care to write only one rule in such cases. You may think that this mistake is not serious as, after all, the association will soon be deleted. But this is not so, and the bogus association may exist indefinitely. If in the example described above the association was derived using two rules, then the database is upgraded and its OracleHome property gets changed. The association with the old home should be removed, but this will not happen until the other rule is fired. However, nothing about the Oracle Home target has changed, so its rule is not triggered and the association remains. Indeed, it is often the case that only one target is changed and the other remains unchanged for a long period of time.

As a general rule of thumb, associations based on data from a specific set of tables should be derived using a single rule with multiple triggers.

Unless there are different reasons for asserting an association exists, you should only use one rule. In such cases, the associations returned by derivation rules should be disjoint. Another way to state this is that for those associations, the set of all rows returned by all rule queries must specify no duplications. An association is identified by source, destination, and association type. So this means that the combination of these three values should be unique.

# Creating Associations for Composite and System Target Types

There are several types of associations that must be considered when constructing either a composite target or a system target, and there are several ways in which these associations can be added to Enterprise Manager. The following describes each of these types of associations, how they are used by the Enterprise Manager framework and the typical approach to how they are created.

## Composite Membership and the Containment Association

The first important association type is the "contains" association. This association type is typically added between a composite target and each of its members. The presence of this association is necessary in order to populate the target navigator (tree) for a composite target. The set of targets that are members of a composite (associated with it through a "contains" association) can be retrieved using the getCompositeMembers() method of the Target class.

These containment associations are most often created during discovery, using either a fully automated approach or through the guided discovery process. In either case, the discovery scripts provided with the plug-in are used to identify the set of containment associations between the composite and its members. Other non-containment associations may also be discovered; however, they will not affect the membership of the composite and will not affect the contents of the target navigator.

If the composite target is also to be treated as a system, it is strongly recommended that the system stencil include rule paths that represent the type of containment associations created in this way. This ensures that the target navigator and system membership display member targets consistently.

## Other Non-Composite Associations (Composite Topology)

In addition to the discovery of containment associations, you may wish to represent other types of associations between the members of your composite topology. These associations may have meaning only to your target administrators and may be used by your plug-in code to perform other operations.

These associations are typically created using a discovery script, either as part of fully automated discovery or through the guided discovery process.

## System Membership Associations

System membership is constructed by the Enterprise Manager framework based on the system stencil. This stencil defines the set of native associations that should be considered when identifying the system members. These native associations are typically either containment or other non-composite associations.

If these associations are found during the evaluation of the system stencil, the destination targets are added as system members.

# Associations to External Targets

Up to this point all of the associations discussed have been between targets that are assumed to be part of the same plug-in, and therefore your plug-in code including discovery and UI has intimate direct knowledge of the configuration and topology of these targets.

However, in some cases your target configuration may include associations to other targets not included in your plug-in, such as an Oracle Database used as an application or backing store for your targets. The configuration of your target knows something about the database that it uses, likely some connection related details such as host-port-sid or host-service.

You would like to represent this association between your target and the database in Enterprise Manager so that if Enterprise Manager is managing the database, the end-user can see this relationship and traverse it to obtain other information about that database and manage it (if appropriate and allowed).

Because you do not know if Enterprise Manager is managing the database and the identifying information you have is not the Enterprise Manager database target name, but instead the connect information, you can construct a derivation rule that maps the connection information in your target's configuration to that of a database in Enterprise Manager.

# Regarding the Timing of Association Creation

Because the creation and deletion of targets and associations can be initiated from various sources (such as automated discovery, guided discovery, derived association rules, and system stencil rules), there are cases where the topology of a composite entity in Enterprise Manager may not appear in sync with the reality of that entity. As a plug-in developer, it is important that you are aware of this and account for it in information you present to end-users whenever possible.

One typical situation that may occur is that the discovery of targets occurs, configuration information is collected, and this is followed by the modification of associations as derivation rules are processed by the Enterprise Manager association framework. In this scenario, the user will first see the topology of the entity, including any association added during discovery. However, the additional associations created by derivation rules will not appear until sometime later.

# Frequently Asked Questions

This section addresses three of the most frequently asked questions:

1. Which tables can I reference in a rule query?
2. Are there guidelines for when to use target properties?
3. What is the relationship between discovered and derived associations?

# Which tables can I reference in a rule query?

In most cases, your query will just reference configuration (Enterprise Configuration Management) tables using the CM$ views, and so will your triggers. For a more complete list of objects that can be referenced, see Using Association Derivation

Rules Syntax and Semantics.If you refer to other tables and if that data may change independently of Enterprise Configuration Management table changes, then the associations may not be updated when needed. If you have a use case in which a non Enterprise Configuration Management table is referenced where changes to that table must trigger rule evaluation, contact your Oracle representative.Another consideration is the component in which the table is located. If the table your rule references is not part of the Enterprise Manager core EDK, your plug-in must account for the dependency on that table's plug-in. For example, you must ensure that any object that you reference already exists in the repository using a plug-in dependency mechanism.

## Are there guidelines for when to use target properties?

Target properties are being treated as configuration data and there is an Enterprise Configuration Management snapshot table that is populated for each target type. Some care should be taken in using data from this table:

- Many target properties are set at discovery time and never modified.

- Querying name/value pair data can be awkward and take longer than queries on other tables where the data is more structured.

  – If the data is available from both the target properties table and an Enterprise Configuration Management snapshot table, you should use the latter.

  – If you need to add collection of configuration data, you should do so in an Enterprise Configuration Management table, not as a new row in the target properties table.

In general, the use of target properties should be avoided and data should be collected and modelled using standard ECM mechanisms.

However, a rule may need to refer to target properties if, for example, the target has no Enterprise Configuration Management collections that can be added. If an association to such a target is to be created, there must be some way to identify it (for example, the rule must refer to its target properties).

If you must use target properties, you should reference `MGMT_TARGET_PROPERTIES` in your rule query. You can also reference `MGMT$TARGET_PROPERTIES` in the rule query if the view already performs the join you need to do. However, in triggers you must use the `GC$TARGET_PROPERTIES` view for the `orcl_tp_config` snapshot type.

`MGMT_TARGET_PROPERTIES` should be used in queries because it is more efficient, but may include some properties not available in the `GC$` view. Triggering is only available based on property changes in the `GC$` view. For example, the `GC$` view only includes those properties that are properly registered with Enterprise Manager.

## What is the relationship between discovered and derived associations?

This is another example of overlapping associations (for more information, see Using Overlapping Associations ). For instance, you may have discovery logic that discovers an association between targets T1 and T2, plus a rule that derives the same association. Oracle recommends that you do not write two sets of logic to create the same association. In this case it is suggested that:

- If a derivation rule is needed because the association may change, you should just write the derivation rule.

- If the association that is discovered will not change until the source or destination is removed, then discovering the association is fine and may be simpler or more efficient.

If you do write two sets of logic to create the same association (discovery logic and derivation rule), then the discovered association will remain and the derivation logic will also assert the existence of that association. If the rule evaluation later determines that the association should no longer exist, the rule's assertion will be removed, but the association will continue to exist unless you manually delete the discovered association.

# 12

# Defining Target Discovery

The discovery of targets in Enterprise Manager can be accomplished in several different ways including automated discovery scripts, manual addition of targets by specifying target properties, and manual addition of targets using guided discovery.

Automatic target discovery is the process by which targets are located and added to Enterprise Manager. Automatic discovery begins when the Oracle Management Agent starts up after installation. Targets located on the server where the Management Agent is running are discovered and sent to the Management Repository as targets that are not yet managed. The end user can choose which targets to monitor by promoting these targets as targets managed by Enterprise Manager.

This chapter contains the following sections:

- Introduction to Defining Target Discovery
- Creating Discovery XML
- Creating the Discovery Script
- Packaging Discovery XML and Discovery Content
- Setting Up and Testing Discovery
- Manually Adding Targets
- Configuring and Promoting Targets for Monitoring by Enterprise Manager
- Examples for Using Generic Discovery Framework
- Configuring Automatic Discovery For Plug-ins

## Introduction to Defining Target Discovery

As a plug-in developer, you are responsible for the following steps within the discovery process:

1. Create discovery metadata.

   Use the Discovery XML Schema Definition (XSD) for guidelines about creating a discovery metadata XML file.

   Within the metadata:

   - Define the discovery modules using the `DiscoveryModule` element.
   - Define discovery parameters (if required) using the `DiscoveryInput` element.

   For information about creating discovery metadata, see Creating Discovery XML.

2. Create the discovery script using Perl.

   The discovery script enables the Management Agent to automatically discover all the target types belonging to a plug-in.

   For information about creating the discovery script, see Creating the Discovery Script.

3. Identify additional Perl modules or JAR files that are required for discovery.

4. Bundle the discovery metadata and contents into the plug-in staging directory (*plugin_stage*).

   a. Save the discovery XML in the *plugin_stage*/oms/metadata/discovery directory.

   b. Save the discovery content in the *plugin_stage*/discovery directory

   For information about packaging, see Packaging Discovery XML and Discovery Content.

5. Repackage (if necessary) and deploy the plug-in.

   After the plug-in archive is created and the plug-in is deployed to the Management Server, the end user can initiate discovery using the discovery configuration UI for the discovery modules that are registered.

   For information about deploying the plug-in, see Validating, Packaging, and Deploying the Plug-in .

6. Configure automatic discovery:

   a. Log in to Enterprise Manager.

   b. Select **Setup**, then select **Add Target**, and then select **Configure Auto Discovery**.

      The Configure Auto Discovery page appears. For information about the options on this page, see the Enterprise Manager online help.

   For more information about configuring automatic discovery, see Configuring Automatic Discovery For Plug-ins.

7. Test discovery results.

   For information about testing discovery, see Setting Up and Testing Discovery.

# Creating Discovery XML

Oracle provides a Discovery XSD so you can write discovery metadata XML to register with the discovery framework. Registering with discovery framework enables the discovery pages to launch discovery of the target types belonging to a plug-in.

For more information about the Discovery XSD, see the Extensibility Development Kit (EDK) specifications.

The following section and the Discovery Integration XML With Discovery Parameters example provide examples of discovery XML.

## Generic Discovery Integration Example

In this example, discovery requires no information entered and promotion of the target does not require any special logic.

There are no special requirements for configuring the target except that you must have access to the UI.

The following example provides the discovery integration XML for this discovery.

> **✏️ Note:**
>
> You are not restricted on the naming of the discovery XML file. However, the standard convention is `plugin_discovery.xml`.

**Example: Discovery Integration XML**

```
<?xml version="1.0" encoding="UTF-8"?>
        <EmTargetDiscovery
            xmlns="http://www.oracle.com/EnterpriseGridControl/disc_metadata"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/
disc_metadata discovery.xsd ">
            <DiscoveryInfo>
                <AutomaticDiscovery>
                    <DiscoveryModule name="simple_disc_plugin"
resourceBundlePkg="oracle.sysman.simpleplugin.rsc.simplePluinMsg">
            <Display NLSID="SIMPLE_DISC_MODULE">
                <NlsValue>simple_disc_plugin</NlsValue>
            </Display>

                    <SupportedAgentOsList>
                        <SupportedAgentOs>2000</SupportedAgentOs>
                    </SupportedAgentOsList>
        <BasicDiscoveryInfo>
          <DiscoveryScript>SimplePluginDisc.pl</DiscoveryScript>
          <DiscoveryCategory>SIMPLE_PLUGIN_DISC</DiscoveryCategory>
        </BasicDiscoveryInfo>
                <TypesDiscovered>
                    <TargetType>simple_plugin_target_type1</TargetType>
                    <TargetType>simple_plugin_target_type2</TargetType>
                </TypesDiscovered>
      </DiscoveryModule>
     </AutomaticDiscovery>
    </DiscoveryInfo>
  </EmTargetDiscovery>
```

After the previous XML is registered, the discovery framework can launch discovery of simplePluginType using the SimplePluginDisc.pl discovery script.

## Discovery Script Example

For example, you can have the following content in the SimplePluginDisc.pl script to discover two target instances:

```
#all the discovery scripts get emdRoot and hostname of the agent as arguments to
the script.
my ($emdRoot, $hostName) = @ARGV;
#Discovery root is sent to the script as env variable.
my $discovery_root = $ENV{DISC_ROOT};
print "\<Targets\>\n";
print  "\<Target TYPE=\"simple_plugin_target_type1\" NAME=\"smpl_tgt1\"\>\n";
print  "\<Property  NAME=\"Prop1\"   VALUE=\"prop1_foo\"/\>\n";
print "     \<Property NAME=\"Prop2\" VALUE=\"prop2_value_bar\"/\>\n";
print "    \<\/Target\>\n";
print "  \<Target TYPE=\"simple_plugin_target_type2\" NAME=\"smpl_tgt2\"\>\n";
print "     \<Property  NAME=\"Prop1\"    VALUE=\"value_foo\"/\>\n";
```

```
print "     \<Property NAME=\"Prop2\" VALUE=\"value_bar\"/\>\n";
print "  \<\/Target\>\n";
print  "\<\/Targets\>\n";
```

This script produces the following output:

```
<Targets>
        <Target type ="simple_plugin_target_type1" NAME="smpl_tgt1" >
           <Property NAME="Prop1"  VALUE="prop1_foo" />
            <Property NAME="Prop2" VALUE="prop2_value_bar" />
         </Target>
        <Target type="simple_plugin_target_type2" NAME="smpl_tgt2" >
             <Property NAME="Prop1" VALUE="value_foo" />
             <Property  NAME="Prop2" VALUE="value_bar" />
        </Target>
 </Targets>
```

After your plug-in is deployed on the Management Server, the discovery module is listed in the Discovery UI. Users can configure discovery of this module on one or more Management Agents. This causes the discovery content of the plug-in to be deployed on the Management Agent and subsequently causing the discovery script to be run at the Management Agent.

For example, when you run this discovery script through autodiscovery, the targets will be generated and sent to the Management Server as Not Yet Managed targets. Using the Discovery Results UI, you can promote these two targets as managed targets by Enterprise Manager.

> **Note:**
>
> There are methods for writing debug information such as EMD_PERL_INFO, EMD_PERL_DEBUG, and EMD_PERL_ERROR, which can be accessed through the Perl package emdcommon.pm. You can then find the trace information (written through Perl methods) in the Oracle Management Agent trace file (emagent_perl.trc) in the Management Agent log directory.

## Overview of the Discovery Metadata Elements

Table 12-1 describes the key elements that define the discovery metadata:

**Table 12-1    Key Elements in a plugin_discovery.xml File**

| Element | Description |
| --- | --- |
| EmTargetDiscovery | The root element for the file |
| DiscoveryInfo | Specifies one or more autodiscovery modules for this plug-in. Each auto discovery module is associated with a discovery script that is run on the Management Agent |
| AutomaticDiscovery | Specifies the autodiscovery module |
| DiscoveryModule | This is an element within the autodiscovery module. It includes the name attribute, which defines the name of the discovery module. |

**Table 12-1    (Cont.) Key Elements in a plugin_discovery.xml File**

| Element | Description |
|---|---|
| SupportedAgentOSList | Specifies the list of Management Agent platforms on which this discovery is supported. It includes the SupportedAgentOS attribute, which defines the platforms. The valid value is 2000 (All platforms). |
| BasicDiscoveryInfo | Specifies the discovery script to be run and an optional category name. It includes the following attributes:<br>• DiscoveryScript: Defines the name of the discovery script<br>• DiscoveryCategory: Defines the category name (optional) |
| TypesDiscovered | Specifies the list of target types that can be discovered using this discovery module. It includes the TargetType attribute, which defines the target type name. |
| DiscoveryInput | Specifies the information to be entered by the user during discovery. The information entered by the user is available as environment variables in the discovery script running at the Management Agent |

# Creating the Discovery Script

After the discovery XML is registered, the discovery framework uses a discovery script to launch discovery.

To create a discovery script:

1. Use Perl to write the top-level discovery script. This script can call Java, Shell, and so on.

2. By default, three variables are provided to the discovery Perl script:

   a. The discovery framework provides *emdRoot* and the host name of the Management Agent as arguments to the script:

      For example:

      ```
      my ($emdRoot, $hostName) = @ARGV;
      ```

   b. The discovery root directory is sent to the script as an environment variable.

      For example:

      ```
      my $discovery_root = $ENV{DISC_ROOT};
      ```

   c. If there are discovery inputs, then they are made available as environment variables.

      For example:

      ```
      my $crs_home = $ENV{'CRS_HOME'};
      ```

Ideally, the output of the discovery script returns all the name-value pairs that are required to add a target of a target type. If not, the user must provide any remaining values at target promotion time.

The Sample Discovery Script example provides an example of a discovery Perl script.

## Discovered Targets DTD

The discovery Perl script must produce output which conforms to the following Document Type Definition (DTD):

**Example: Discovered Targets DTD**

```
<!ELEMENT Targets (Target*) >
<!ATTLIST Targets
 >

<!--

    Target defines a target instance, it may also define Properties

    TYPE(required)      : the target type.
    NAME(required)      : the target name.  It must be unique within a type
                          across all nodes.

    DISPLAY_NAME(optional)    : the display name for a target.will be
                                  defaulted to NAME if not given.
-->

<!ELEMENT Target (Property*) >
<!ATTLIST Target
        TYPE                CDATA #REQUIRED
        NAME                CDATA #REQUIRED
        DISPLAY_NAME        CDATA #IMPLIED
 >

<!--
    A Property tag describes a name-value pair of target instance properties

    NAME(required)      : the property name .
    VALUE(required)     : the property value.
-->
<!ELEMENT Property EMPTY>
<!ATTLIST Property
        NAME CDATA #REQUIRED
        VALUE CDATA #REQUIRED
 >
```

# Packaging Discovery XML and Discovery Content

The following sections describe where to include the discovery XML and content in the plug-in staging directory.

## Location of the Discovery Metadata File

When you complete the discovery metadata XML file, include the file in the following directory of the plug-in staging directory:

*plugin_stage*/oms/metadata/discovery

# Package Discovery Content

Discovery content refers to all the Perl scripts, Perl modules, and JAR files (if any) that are required to perform discovery of a particular target type

For Enterprise Manager Cloud Control 12*c*, discovery content is shipped to the Management Agent only when the user attempts discovery for the first time. Discovery content corresponding to a particular discovery module will reside in its own area.

Discovery and monitoring scripts are separate. Both are parts of your plug-in. You can run discovery without monitoring content. The lifecycle of both are managed by the discovery or plug-in lifecycle frameworks and is transparent to plug-in developers.

You must package the discovery content required for discovering a particular target type. For example, for an existing database discovery, the discovery content is oracledb.pl along with any required utilities for running database discovery.

> **✎ Note:**
>
> You must package the discovery metadata file with the Oracle Management Server archive and *not* the Management Agent archive.

Create a discovery directory under *plugin_stage* for installing content for each discovery plug-in. For more information about the directories under *plugin_stage*, see Validating, Packaging, and Deploying the Plug-in .

**Example: Directory Structure for Installing Discovery Content**

```
plugin_stage/discovery/
                    |
                    |
                    |__other subdirectories created as you specified
```

For Oracle Database discovery, the discovery content might look similar to the following:

```
plugin_stage/discovery/
                    |
                    |
                    |___oracledb.pl
                         |
                         |
                         |__utl
                              |
                              |___oracledbUtl.pl
                                   |
                                   |___initParamFileUtl.pl
                                   |
                                   |___winRegistry.pl
```

If any custom Perl modules are required for the discovery process, then place the modules under a similar directory structure as shown previously. The discovery root variable provided to the discovery script can be used to load this Perl module. For example, if your perl modules are placed under the *plugin_stage*/discovery/utl/pm directory, then you can load them from the discovery script as follows:

```
my ($emdRoot,$hostName,$crsHome) = @ARGV;
my $discovery_root = $ENV{DISC_ROOT};
require "$discovery_root/utl/pm/propertiesFileParser.pm";
require "$discovery_root/utl/pm/Targets.pm";
```

> **Note:**
>
> Perl content that will be used by the discovery module and for other purposes, such as administration of the targets, should be packaged with the discovery bundle as well as the plug-in bundle.

## Java Content Required by Discovery Scripts

Some discovery scripts (such as for Oracle Fusion Middleware) use JAR files in the process of discovery. If the JAR files are discovery-specific only, then they should be in discovery area.

If there are Java methods written to perform discovery, then they should be moved to a separate class file where possible, to avoid shipping content not required by discovery.

Oracle recommends separating discovery content and management content so that discovery can be performed independent of the management content present. This helps you to decide whether you want to install the plug-in to manage the discovered targets, if any. The discovery content should be lightweight and include the files necessary for discovery only.

For example, you can place the JAR containing discovery-specific code of the particular discovery module under the following directory:

```
plugin_stage/discovery/lib
```

> **Note:**
>
> The individual discovery script is responsible for constructing the class path.

You can create your own directory structure in the discovery content area for a particular discovery module. The top level Perl scripts responsible for each discovery module, such as Fusion Middleware, construct the class path before running the Java utilities that they use. Because the JAR files specific to a discovery module will be in their own discovery content area, the discovery Perl script can construct the required class path easily. Again, the code responsible for performing discovery only should be separated out and installed in the discovery content area specific to the particular discovery module.

# Setting Up and Testing Discovery

For testing purposes, you can run discovery:

1.  Log in to Enterprise Manager.

    ```
    https://em_host:em_port/em/
    ```

2. From the console, select **Setup**, then **Add Target**, and then **Configure Auto Discovery**.

   The Configure Auto Discovery page appears.

3. In the Configure Auto Discovery tables, under Discovery Module, select **Multiple Target-Type Discovery on Single Host**.

   The Target Discovery (Agent Based) page appears.

4. Select the required agent host name and click **Run Discovery Now**.

   When target discovery is complete, a Completed Successfully window appears.

# Manually Adding Targets

In addition to automatic discovery, Cloud Control allows you to manually add hosts as well as a wide variety of Oracle software and components as managed targets. When you add a target manually, you do not need to go through the process of discovery by adding the target directly. Discovering targets in this way eliminates the need to consume resources on the agent to perform discovery when it is not needed.

You must be able to specify the properties of a target to be managed and create an Enterprise Manager managed target.

Not all target types can be manually added. During registration with the discovery framework, the target type owner indicates whether a target type can be manually added or not.

See the following sections for instructions:

- Manually Adding Host Targets
- Manually Adding Non-Host Targets

## Manually Adding Host Targets

A wizard guides you through the process of manually deploying a Management Agent to a new host target.

For instructions on installing a Management Agent, see "Installing Oracle Management Agent" in the *Enterprise Manager Cloud Control Basic Installation Guide*.

## Manually Adding Non-Host Targets

A configuration page or wizard based on target type metadata listing all the instance properties required to manage target is displayed.

You can specify a name for the target and provide the required configuration information.

To add targets manually to Enterprise Manager:

1. Log in into Enterprise Manager.

2. From the **Setup** menu, select **Add Target**, then select **Add Targets Manually**.

   Enterprise Manager displays the Add Targets Manually page.

3. Under the Add Targets Manually page, go to the Add Targets Manually sub-section and select one of the following options:

- Add Non-Host Targets Using Guided Process

  From the Target Types list, select one of the target types to add, such as **Oracle Cluster and High Availability Service**, **Oracle Database Machine**, or **WebLogic Domain Discovery**, then click **Add Using Guided Discovery**. This process will also add related targets.

- Add Non-Host Targets by Specifying Target Monitoring Properties

  From the Target Type list, select one of the target types to add, such as Fusion J2EE Application, Applications Utilities, or Supplier Portal, and from the Monitoring Agent list, select the required Monitoring Agent, then click **Add Manually**.

4. After you select the target type, you will follow a wizard specific to the target type to add the target.

   Upon confirmation, the target becomes a managed target in Enterprise Manager. Enterprise Manager accepts the information, performs validation of the supplied data where possible and starts monitoring the target.

> **Note:**
>
> For more information about adding targets manually to Enterprise Manager, see Oracle Enterprise Manager Online Help.

## Configuring and Promoting Targets for Monitoring by Enterprise Manager

Discovery of targets on Enterprise Manager managed hosts provides you with a list of targets such as new databases, and SQL servers which are not yet managed by Enterprise Manager. This helps you to determine if any of the new targets found are candidates for monitoring and managing by Enterprise Manager.

The Enterprise Manager UI allows you to review discovered unmanaged targets and promote targets to be managed by Enterprise Manager for monitoring.

> **Note:**
>
> For more information about configuring and promoting targets, see Oracle Enterprise Manager Online Help.

## Examples for Using Generic Discovery Framework

You can use the generic discovery UI to launch and schedule discovery to run periodically at the Management Agent. For the generic discovery UI to launch discovery, you must specify the inputs that are required through the registration XML. Enterprise Manager uses this information to construct the generic discovery UI.

# Discovery Integration Example Requiring User Input

In this example, discovery requires the user to enter information. The user is prompted to enter values for CRS_HOME and CRS_HOME1. The user does not have to enter a value for CRS_HOME but a value for CRS_HOME1 is mandatory

There are no special requirements for configuring the target except that you must have access to the UI.

> **Note:**
>
> While supported, Oracle does not recommend the use of required discovery inputs because it eliminates the benefits of automatic discovery. You can use optional discovery inputs as hints to optimize the discovery process.

**Example: Discovery Integration XML With Discovery Parameters**

```
<?xml version="1.0" encoding="UTF-8"?>
    <EmTargetDiscovery
        xmlns="http://www.oracle.com/EnterpriseGridControl/disc_metadata"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xsi:schemaLocation="http://www.oracle.com/EnterpriseGridControl/
disc_metadata discovery.xsd ">
        <DiscoveryInfo>
            <AutomaticDiscovery>
                <DiscoveryModule name="simple_disc_plugin"
resourceBundlePkg="oracle.sysman.simpleplugin.rsc.simplePluinMsg">

             <Display NLSID="SIMPLE_DISC_MODULE_MSG_ID">
                 <NlsValue>simple_disc_plugin</NlsValue>
             </Display>
           <SupportedAgentOsList>
                 <SupportedAgentOs>2000</SupportedAgentOs>
           </SupportedAgentOsList>

           <BasicDiscoveryInfo>
          <DiscoveryScript>simple_plugin_disc.pl</DiscoveryScript>
          <DiscoveryCategory>SIMPLE_PLUGIN_DISC</DiscoveryCategory>
           </BasicDiscoveryInfo>
           <TypesDiscovered>
                 <TargetType>simple_plugin_target_type1</TargetType>
                 <TargetType>simple_plugin_target_type2</TargetType>
           </TypesDiscovered>

           <!-- optional discovery hint -->
           <DiscoveryInput name="CRS_HOME" isRequired="false">
           </DiscoveryInput>

           <!-- a required discovery input -->
           <DiscoveryInput name="CRS_HOME1" isRequired="true">
           </DiscoveryInput>

        </DiscoveryModule>
        </AutomaticDiscovery>
```

```
        </DiscoveryInfo>
    </EmTargetDiscovery>
```

After the previous XML is registered, the discovery framework can launch discovery using the simple_plugin_disc.pl discovery script

The simple_plugin_disc.pl script can access the discovery parameters as illustrated in the sample `simple_plugin_disc.pl` script.

**Example: Sample Discovery Script**

```
my $discovery_root = $ENV{DISC_ROOT};

#Inputs passed to the script.
my $crs_home = $ENV{'CRS_HOME'};
my $crs_home1 = $ENV{'CRS_HOME1'};

#add the logic here to find the targets.

print "\<Targets\>\n";
print " \<Target TYPE=\"simple_plugin_target_type1\" NAME=\"smpl_tgt1\"\>\n";
print "   \<Property  NAME=\"Prop1\"  VALUE=\"prop1_foo\"/\>\n";
print "     \<Property NAME=\"Prop2\" VALUE=\"prop2_value_bar\"/\>\n";
print "   \<\/Target\>\n";
print " \<Target TYPE=\"simple_plugin_target_type2\" NAME=\"smpl_tgt2\"\>\n";
print "   \<Property  NAME=\"Prop1\"  VALUE=\"value_foo\"/\>\n";
print "     \<Property NAME=\"Prop2\" VALUE=\"value_bar\"/\>\n";
print "   \<\/Target\>\n";
print  "\<\/Targets\>\n";
```

> **Note:**
>
> For guidelines about the output of the discovery Perl script, see Discovered Targets DTD.

# Configuring Automatic Discovery For Plug-ins

You can configure automatic discovery for targets in a plug-in to run at the Management Agent-side. Currently, automatic discovery is scheduled to run every day.

Configuration is done from the Oracle Management Server where your metadata resides.

> **Note:**
>
> For plug-ins deployed before the Management Agents are installed or plug-ins deployed to new hosts, discovery runs every 24 hours automatically (only if discovery does not require any user inputs).
>
> You can configure parameters from the Enterprise Manager UI after the Management Agents are installed or deployed.

After discovery, the targets are sent to Enterprise Manager. The end user can then review the targets and choose which targets to monitor by promoting the targets as targets managed by Enterprise Manager.

> **Note:**
>
> For more information about configuring automatic discovery, see Oracle Enterprise Manager Online Help.

# 13
# Adding Compliance Standards

The Oracle Enterprise Manager Compliance Management solution provides the capability to define, customize, and manage compliance frameworks and compliance standards.

To view a visual demonstration about the Compliance Management framework, access the following URL and click **Begin Video**.

```
https://apex.oracle.com/pls/apex/f?
p=44785:24:0::NO:24:P24_CONTENT_ID,P24_PREV_PAGE:5773,1
```

This chapter contains the following sections:

- Introduction to Adding Compliance Standards
- About the Compliance Standard Rules
- Defining Compliance Standards
- Defining a Compliance Framework
- Defining Compliance Content
- Removing Compliance Content
- Supporting Translation
- Packaging Compliance XML
- Setting Up and Testing Compliance Standards and Rules
- More Compliance Examples
- Publishing Compliance Content Using Self Update

## Introduction to Adding Compliance Standards

As a plug-in developer, you are responsible for the following steps when adding compliance standards:

1.  Define compliance standard rules.

    Compliance standard rules can be either of the following:

    - Repository check-based rules
    - Real-time monitoring rules.

    For information about defining compliance standard rules, see About the Compliance Standard Rules.

2.  Define a compliance standard.

    For more information, see Defining Compliance Standards.

3.  Define a compliance framework.

    For more information, see Defining a Compliance Framework.

4. Package the compliance standard rules, standards, and framework as metadata XML.

For more information, see Packaging Compliance XML.

5. Set up and test the compliance content.

For more information, see Setting Up and Testing Compliance Standards and Rules.

6. Deploy the plug-in.

For information about deploying plug-ins, see Validating, Packaging, and Deploying the Plug-in .

## Assumptions and Prerequisites

This chapter assumes you are familiar with the following:

- Plug-in development overview, including how to package a plug-in and its XML files.

- If you are installing compliance data with the `emctl register oms metadata -service gccompliance` command, it will require an `EM_COMPLIANCE_UTIL.trigger_rule_dependency_job` callback. You must enter the following commands through SQL*Plus as the `SYSMAN` user:

```
begin EM_COMPLIANCE_UTIL.trigger_rule_dependency_job;
end;
/
```

> ✏️ **Note:**
>
> This is necessary *only* if you are using the `emctl register oms metadata -service gccompliance` command to install compliance content.
>
> If you are installing the plug-in, then you do not have to enter the previous SQL.

For more information about the `emctl register oms metadata` command, see Updating Deployed Metadata Files Using the Metadata Registration Service (MRS).

# About the Compliance Standard Rules

This section provides a description of the following:

- Defining Repository Check-based Rules
- Defining Real-time Monitoring Rules

## Defining Repository Check-based Rules

A repository check-based rule checks the configuration state of one or more targets. A rule is compliant if the test fails to identify a violation. In other words, the test

determines that the configuration item is in the required state or has the prescribed value. Any rule that uncovers a violation is noncompliant.

The data source that is evaluated by a rules test condition can be based on a repository query. A rules test condition can be implemented using a threshold condition based on the underlying metrics or queries column value, or SQL expression, or a PL/SQL function. (The policies are similar to Oracle Enterprise Manager 10*g* Release 5).

The key points in this area include:

- Defining Compliance Standard Rules, Compliance Standards, and Compliance Frameworks
- Replacing out-of-box policy groups (10.2.*x*/11.10) with Compliance Standards that you create referring to Compliance Standard Rules
- Mapping your compliance standards to the appropriate Compliance Frameworks
- Defining Oracle Business Intelligence Publisher (BI Publisher) reports for compliance

The following example provides the syntax for defining repository rules and the next example provides an example of a repository rule definition.

> **Note:**
>
> For the complete compliance XML Schema Definitions (XSDs), see the following JAR file:
>
> `$ORACLE_HOME/sysman/jlib/gccomplianceCommon.jar`

> **Note:**
>
> For additional examples, see More Compliance Examples.

**Example: Repository Rule Definition Syntax**

```
<xsd:complexType name="RuleT">
    <xsd:sequence>
      <xsd:element name="DisplayName" type="std:DisplayString256Def"
minOccurs="0"/>
      <xsd:element name="TargetType" type="std:Name256Def"/>
      <xsd:element name="IsSystem" type="std:BooleanDef" minOccurs="0"
default="false"/>
      <xsd:element name="IsHidden" type="std:BooleanDef" minOccurs="0"
default="false"/>
      <xsd:element name="evaluateAlways" type="std:BooleanDef"
default="false" minOccurs="0"/>
      <!-- E.g. target version, platform based filter -->
      <xsd:element ref="std:TargetPropertyFilter" minOccurs="0"/>
      <xsd:element name="Description" type="std:DisplayString800Def"
```

```
minOccurs="0"/>
      <xsd:element name="Impact" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="Recommendation" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="FixLinkList" type="std:FixLinkListT"
minOccurs="0"/>
      <xsd:element name="ViolationContextList"
type="std:ViolationContextListT"/>
      <xsd:element name="CheckSource" type="std:CheckSourceT"
minOccurs="1" maxOccurs="1"/>

      <xsd:element name="Severity" default="MinorWarning" minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="MinorWarning"/>
            <xsd:enumeration value="Warning"/>
            <xsd:enumeration value="Critical"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="LifeCycleStatus" default="Development"
minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Development"/>
            <xsd:enumeration value="Production"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="KeywordList" type="std:KeywordListT"
minOccurs="0"/>
      <xsd:element name="UrlLink" type="std:String4000Def"
minOccurs="0"/>
      <xsd:element name="ViolationMessage"
type="std:DisplayString800Def" minOccurs="0"/>
      <xsd:element name="ClearViolationMessage"
type="std:DisplayString800Def" minOccurs="0"/>
      <xsd:element name="Author" type="std:Name256Def" minOccurs="0"/>
      <xsd:element name="LastUpdatedBy" type="std:Name256Def"
minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="std:Name256Def" use="required"/>
    <xsd:attribute name="oms_version" type="std:Name32Def"
use="required"/>
  </xsd:complexType>
```

The following example is defined for oracle_database target_type, which is part of the database plug-in.

Youcan define a rule for any target type registered with Enterprise Manager.

**Example: Sample Rule**

```
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_rule1">
    <DisplayName nlsid="SAMPLE_RULE_1_NAME">Sample Rule 1</DisplayName>
    <TargetType>oracle_database</TargetType>
    <IsSystem>true</IsSystem>
    <TargetPropertyFilter>
        <PropertyItem>
            <PropertyName>orcl_gtp_operating_system</PropertyName>
            <ValueList>
                <Value>Windows</Value>
            </ValueList>
        </PropertyItem>
        <PropertyItem>
            <PropertyName>orcl_gtp_target_version</PropertyName>
            <ValueList>
                <Value>8.1.6+</Value>
            </ValueList>
        </PropertyItem>
    </TargetPropertyFilter>
    <Description nlsid="SAMPLE_RULE_1_DESC">Checks for use of a single
control file</Description>
    <Impact nlsid="SAMPLE_RULE_1_IMPACT">The control file is one of the
most
important files in an Oracle database. It maintains many physical
characteristics
and important recovery information about the database. If you lose the
only copy
of the control file due to a media error, there will be unnecessary
down time and
other risks.</Impact>
    <Recommendation nlsid="SAMPLE_RULE_1_RECO">Use at least two control
files that are multiplexed on different disks.</Recommendation>
    <ViolationContextList>
        <Column type="String" name="FILE_LIST">
            <DisplayLabel nlsid="SAMPLE_RULE_1_COL_1">FILE_LIST</
DisplayLabel>
            <IsHidden>false</IsHidden>
            <IsKey>false</IsKey>
        </Column>
        <Column type="Number" name="CONTROL_FILE_COUNT">
            <DisplayLabel
nlsid="SAMPLE_RULE_1_COL_2">CONTROL_FILE_COUNT</DisplayLabel>
            <IsHidden>false</IsHidden>
            <IsKey>false</IsKey>
        </Column>
    </ViolationContextList>
    <CheckSource>
        <RepositoryCheckDefinition>
            <Metric>
    <TargetType>oracle_database</TargetType>
    <MetricName>sample_rule1</MetricName>
    <SourceType>SQL</SourceType>
        <Source>select CONTROL_FILE_COUNT, FILE_LIST, TARGET_GUID from
```

```
MGMT$CS_DB_CONTROL_FILE_COUNT</Source>
    <MetricColumnList>
        <MetricColumnInfo>
                <ColumnName>FILE_LIST</ColumnName>
                <ColumnType>String</ColumnType>
                <isKey>false</isKey>
                <ColumnLabel nlsid="SAMPLE_RULE_1_COL_1">FILE_LIST</
ColumnLabel>
                    </MetricColumnInfo>
        <MetricColumnInfo>
                <ColumnName>CONTROL_FILE_COUNT</ColumnName>
                <ColumnType>Number</ColumnType>
                <isKey>false</isKey>
                <ColumnLabel
nlsid="SAMPLE_RULE_1_COL_2">CONTROL_FILE_COUNT</ColumnLabel>
        </MetricColumnInfo>
    </MetricColumnList>
            </Metric>
        <ParameterList>
            <RuleParameter>
                <ParamName>CONTROL_FILE_COUNT</ParamName>
                <ParamType>Number</ParamType>
            </RuleParameter>
        </ParameterList>
        <ParameterDefaultSettings>
            <ParamValue>
                <ParamName>CONTROL_FILE_COUNT</ParamName>
                <MinorWarnThreshold>1</MinorWarnThreshold>
            </ParamValue>
        </ParameterDefaultSettings>
        <TestCondition>
            <ThresholdCriteria>
                <ColumnName>CONTROL_FILE_COUNT</ColumnName>
                <TestOperator>EQ</TestOperator>
                <ThresholdValue>1</ThresholdValue>
                <ThresholdType>Number</ThresholdType>
            </ThresholdCriteria>
        </TestCondition>
        </RepositoryCheckDefinition>
    </CheckSource>
  <Severity>MinorWarning</Severity>
  <LifeCycleStatus>Production</LifeCycleStatus>
  <KeywordList>
      <Keyword nlsid="CONFIGURATION">Configuration</Keyword>
  </KeywordList>
  <ViolationMessage nlsid="SAMPLE_RULE_1_VIOL_MSG">The database has
an insufficient number of control files.</ViolationMessage>
  <ClearViolationMessage nlsid="SAMPLE_RULE_1_VIOL_CLEAR_MSG">The
database has sufficient number of control files.</ClearViolationMessage>
  <Author>SYSMAN</Author>
</Rule>
```

Table 13-1 provides a description of the tags used to define a repository rule:

**Table 13-1    Key Tags for Defining Repository Rules**

| Tag | Subtag | Description |
| --- | --- | --- |
| DisplayName | NA | The display name of the rule. It provides the nlsid attribute to support the translation of messages. |
| TargetType | NA | The type of target to which this rule is can be associated |
| IsSystem | NA | True for out-of-the-box rules. Otherwise, False |
| IsHidden | NA | False by default. When set to True, the IsHidden rules are not visible in the UI and no events are generated. This element should be set to true for out-of-the-box rules |
| Description | NA | The description of the rule. It provides the nlsid attribute to support the translation of messages. |
| Impact | NA | Impact if the rule violates (when rule is noncompliant). It provides the nlsid attribute to support the translation of messages |
| ViolationContextList | NA | Violation context defines a violation to a rule uniquely. Violation context lists columns from <Source> Query, which will be visible as a part of the violation. Each column must mark as key or non-key. The mandatory target_guid column from <Source> query is implicitly added to the violation context and should not be included in the violation context explicitly. |
| NA | Column | Metric Column name. Uses Attributes name and type<br>• DisplayLabel: Display name for column. It provides the nlsid attribute to support the translation of messages<br>• IsHidden: True, if this column should not be displayed as a part of a violation context. Otherwise, False.<br>• IsKey: True, if this column is a key |
| CheckSource | NA | Defines the data source for Rule evaluation. |
| ..RepositoryCheckDefinition | NA | Defines data source for a repository rule. |

**Table 13-1    (Cont.) Key Tags for Defining Repository Rules**

| Tag | Subtag | Description |
| --- | --- | --- |
| NA | Metric | Defines data source query.<br><br>• `MetricName`: Name of metric<br>• `SourceType`: SQL. The only supported SourceType<br>• `Source`: This is a SQL query written over MGMT$_% views. (Enterprise Manager provides MGMT$_% views, see *Oracle Enterprise Manager Cloud Control Management Repository Views Reference*.). If required, this SQL query can be written over other provided views that they have a direct SELECT privilege to the MGMT_VIEW user in Enterprise Manager.<br><br>**Note**: You can specify target_guid within a rule source query. This ensures that target_guid will get bound in the query at runtime. This can lead to improved performance.<br><br>For example:<br><br>```select a.cm_target_guid as target_guid, a.SESS_LAZY_DESER_ENABLED from MGMT$WEBLOGIC_CLUSTER a, mgmt$target_flat_members mtfm where mtfm.member_target_guid = a.cm_target_guid and mtfm.aggregate_target_type = 'exalogic_system' and a.cm_target_guid = :target_guid```<br><br>• `MetricColumnList`: List of columns in Source query<br>• `MetricColumnInfo`: Metric column<br>`ColumnName`: Metric Column name<br>`ColumnType`: Metric Column Type<br>`isKey`: True, if column is a key column. Otherwise, False.<br>`ColumnLabel`: Column display name |
| ParameterList | NA | List of parameters<br><br>**Note**: If parameters are specified and used in a where clause, then you can customize the parameter value at compliance standard rule and target type level or compliance standard rule and target instance level. This enables the user to customize or control the check definition behavior per target instance or at the target type level. |
| RuleParameter | NA | Parameter definition |
| NA | ParamName | Name of parameter |

**Table 13-1    (Cont.) Key Tags for Defining Repository Rules**

| Tag | Subtag | Description |
| --- | --- | --- |
| `NA` | `ParamType` | parameter Type (String or number) |
| `ParameterDefaultSettings` | | Default values for parameters |
| `NA` | `ParamValue` | Define a default value for a parameter:<br><br>• `ParamName`: Name of parameter<br>• `CritThreshold/WarnThreshold/ MinorWarnThreshold`: Parameter default value. For critical severity, use `CritThreshold`. For warning severity, use `WarnThreshold`. For minor warning severity, use `MinorWarnThreshold`. |
| `TestCondition` | `NA` | The TestCondition tag operates over the data source fetched by running the Metric's <Source> Query. Any data source row that satisfies the condition is a violation to the rule |
| `NA` | `ThresholdCriteria` | Requires a column from Source Query, a threshold value, and operator (=,<,>, and so on) to relate the column value and threshold value. |
| `NA` | `SqlWhereClauseCriteria` | Requires a SQL condition over the columns from <Source> query. Optionally, this condition can include one or more parameters. |
| `Severity` | `NA` | Severity of Rule (Critical, Warning, MinorWarning) |
| `LifeCycleStatus` | `NA` | Lifecycle status of rule, (Development or Production) |
| `UrlLink` | `NA` | Detail URL for the Rule, containing details about the rule |
| `ViolationMessage` | `NA` | Message recorded with violation (for rule). Used for notifications. It provides the nlsid attribute to support the translation of messages |
| `ClearViolationMessage` | `NA` | Message recorded with clearing of violation (for rule). Used for notifications. It provides the nlsid attribute to support the translation of messages. |
| `KeywordList` | `NA` | List of keywords associated with the Rule. |
| `NA` | Keyword | Keywords applicable to the compliance standard |
| `Author` | `NA` | Rule Author. |

## Defining Real-time Monitoring Rules

You can use a real-time monitoring rule to monitor any action that can happen against a file, a database object, or a Microsoft Windows Registry key. It can also monitor the starting and stopping of processes, and the logging in, logging out, and switching user

(`su`) activity of users. The real-time aspect of the monitoring means that it captures the exact time the action occurred along with the user that performed the action.

Results from real-time monitoring can be reconciled with a Change Management system such as BMC Remedy. This reconciliation can automatically determine if an action was supposed to happen (authorized) or not (unauthorized). If a customer does not have a Change Management server, this audit status annotation can be made manually through the UI.

A major part of any IT compliance initiative is to ensure that your IT operations staff are making changes and managing the environment according to corporate policies. By reconciling what is happening in the environment to the customer's change management process, real-time monitoring helps to identify out-of-policy actions that will either lead to a high risk environment, or a compliance control that will fail audits.

Creating a real-time monitoring rule involves the following steps. These are explained further below.

- Choose the target type and entity type being monitored. A rule can also be limited based on certain target type properties (OS, target version, hardware platform, and target lifecycle)

- Choose one or more target type facets to monitor

- Choose one or more observations to watch for

- Choose zero or more facets to filter the results that are monitored

- Choose the change management reconciliation options

Integration points in this area include:

- Defining (one or more) facets for your target types. Facets define the low level artifacts that will be monitored

- Defining new compliance standard rules for new or existing compliance standards

- Mapping your compliance standard rules and compliance standards to the out-of-box compliance frameworks that are related to industry standard frameworks.

- Creating connectors to support new ticketing systems (including definition of custom region). This can also be used to extend out-of-box change reconciliation support (currently limited to Remedy 7.*x*). For information about the process for creating new connectors, see the *Oracle Enterprise Manager Connector Integration Guide*.

## What Entity Types Can I Monitor?

When you define a real-time monitoring rule, the first thing you have to decide is what entity type on a host to monitor. For Oracle Enterprise Manager Cloud Control 12*c*, the following entity types can be monitored with Real-time Monitoring Rules:

- OS File

- OS Process

- OS User

- Microsoft Windows Registry

- Microsoft Active Directory User

- Microsoft Active Directory Computer

- Microsoft Active Directory Group
- Oracle Database Table
- Oracle Database View
- Oracle Database Procedure
- Oracle Database User
- Oracle Database Index
- Oracle Database Sequence
- Oracle Database Function
- Oracle Database Package
- Oracle Database Library
- Oracle Database Trigger
- Oracle Database Tablespace
- Oracle Database Materialized View
- Oracle Database Cluster
- Oracle Database Link
- Oracle Database Dimension
- Oracle Database Profile
- Oracle Database Public DB Link
- Oracle Database Synonym
- Oracle Database Public Synonym
- Oracle Database Segment
- Oracle Database Type
- Oracle Database Role
- Oracle Database SQL Query Statement

These entity types are fixed by the capabilities of the current release and cannot be extended. However, you can use them when creating facets and Real-Time monitoring rules.

In addition to facets defining what can be monitored, there is a set of entities that can be used for filtering also. The following list includes the most commonly used filtering entity types:

- OS Process
- OS User
- Oracle Database User
- Time Window
- Host

When you create a Real-time monitoring rule, choose what to monitor (that is, what files). Then choose if you want to use filtering so that only actions performed by certain users, or at certain periods of time are monitored.

# About Real-time Monitoring Facets

Target Type Facets are used to specify the list of entities to monitor. These facets can be used again at a later time in any number of rules. They can be created on their own, or created inline with a Real-time Monitoring rule.

In the case of OS File monitoring, a facet could be a list of distinct single files, patterns with wildcards that would include many files, or simply an entire directory.

These patterns can also include parameters with a default, but can be overridden as required for each target.

The following are some examples of facets that may be defined for a HOST target type and an OS FILE entity type:

**User Credential Files**

- `/etc/passwd`

- `/etc/shadow`

- `/etc/mail/trusted-users`

**Network Configuration Files**

- `/etc/hosts`

- `/etc/resolv.conf`

- `/etc/hosts.*`

- `/etc/defaultrouter`

- `/etc/nsswitch.conf`

- `/etc/netmasks`

  `{app_install_directory}/network/config`

Here are some examples of facets that might be defined for a HOST target type and an OS PROCESS entity type. These might be monitored in real-time because any of these processes started on a production server could lead to a significant security risk.

**Network Configuration Tools**

- `ifconfig`

- `xhost`

The following table provides a list of hypothetical facets that you might create for your given target type. The facet name can be anything you choose. For some plug-in developers, there might be many more facets than these limited examples. For each facet, there is a description of the included patterns.

| Target-Type Facet | Description |
| --- | --- |
| Log files | List each log file the target type has. |
| | Customers want to monitor when regular users modify a log file (not a system user) |

| Target-Type Facet | Description |
|---|---|
| Binary Files | List each binary the target type has. |
| | Rules can be created to monitor if a binary is tampered with or when a binary is patched. Instead of listing each individual binary, it can also list a whole directory, but exclude frequently changing files |
| Library Files | List each library the target type has. |
| | Rules can be created to monitor if a library is tampered with or when a library is patched. Instead of listing each individual library, it can also list a whole directory, but exclude frequently changing files |
| General Configuration Files | List any configuration files that are user changeable normally, but the user might want to capture changing. |
| Security Key Files | List any files that store certificates, keys, and so on. |
| | This can be a whole directory also, but exclude files that change regularly. This is to monitor if any users read the files in an attempt to get the content of the certificates. |
| Security Configuration Files | List any files that configure how security works in the target type, such as encryption configuration, and so on |
| Application Users | List the typical application users (that is, Oracle, root), and so on. |
| | Users can use this facet to filter monitoring changes where they do not care if the application user makes the change |
| Utility Processes | Any utility processes that normally run during a maintenance period, but should not be run during production |
| Registry Keys | Any Microsoft Windows registry keys that affect the configuration of the target |
| Configuration Tables | Any database tables that store configuration data. |

## Creating Real-time Monitoring Facets

This section provides an overview of the XML tags used in creating a real-time monitoring facet and an example of XML fragment showing facet creation. Facets can be created on their own as shown in this example, or inline with a real-time monitoring rule creation.

Table 13-2 provides descriptions of the tags used to define a Real-time monitoring facet:

**Table 13-2    Key Tags Used to Define a Real-Time Monitoring Facet**

| Tag | Subtag | Description |
|---|---|---|
| Name | NA | The internal name of the facet. This must be unique across all facets that exist and is not visible on the UI. |
| DisplayName | NA | The display name of the facet. It provides the nlsid attribute to support the translation of messages. |
| TargetType | NA | The type of target to which this rule can be associated. |

**Table 13-2    (Cont.) Key Tags Used to Define a Real-Time Monitoring Facet**

| Tag | Subtag | Description |
|---|---|---|
| EntityType | NA | The entity type for which you are creating the facet (such as osfile, osprocess, osuser, and so on) |
| IsSystem | NA | True, for out-of-the-box rules. Otherwise, False. |
| Description | NA | The description of the facet. It provides the nlsid attribute to support the translation of messages. |
| Author | NA | The Enterprise Manager user that is the author of the facet. |
| LastUpdatedBy | NA | The Enterprise Manager user that last updated the facet. This should be same as the author for your initially created data. |
| SourcePattern/ GeneralPattern: | NA | Container holding the pattern definition that makes up the facet |
| NA | Patterns/Pattern | Collection of patterns that define the facet. A single facet can be made up of include and exclude patterns.<br><br>• Value: An actual pattern. This pattern can include wildcards and parameters. Parameters are specified or bounded by { and }. Parameters must have a default value which is defined further down in the XML.<br><br>The entity type determines the limitations on how wildcards are used. The product documentation outlines these limitations per entity type.<br><br>• Description: Description of the pattern. It provides the nlsid attribute to support the translation of messages.<br><br>• IsIncluded: Whether this pattern is an include pattern or exclude pattern. The notion of include or exclude is useful for wildcards. You can have a pattern which includes an entire directory, then you can exclude subdirectories or individual files under that included directory. A value of 0 indicates that this pattern is an exclude pattern. 1 indicates an include pattern. |

**Table 13-2    (Cont.) Key Tags Used to Define a Real-Time Monitoring Facet**

| Tag | Subtag | Description |
| --- | --- | --- |
| NA | Parameters/<br>Parameter | Collection of pattern default values for each parameter introduced in the patterns. Parameters are not shared across facets. If you use the same parameter name in two facets, each facet must define its own default value. |
| | | • Name: The parameter name used in the patterns defined above in the XML |
| | | • Value: The default value for this parameter. Users can override this parameter value per target when associating a Compliance Standard to one or more targets where this facet is in use. |
| | | • Description: Description of the parameter. It provides the nlsid attribute to support the translation of messages. |
| | | • isActive: Whether this parameter is currently in use in the list of patterns. This should always be 1 as you would not define new parameters without using them in the patterns |

**Example: Sample Facet Definition**

```
<Facet xmlns="http://www.oracle.com/DataCenter/ConfigStd" is_time_window="0">
    <Name>network_configuration_files</Name>
    <DisplayName nlsid="SAMPLE_FACET_DNAME">Networking configuration files</
DisplayName>
    <TargetType>host</TargetType>
    <EntityType>osfile</EntityType>
    <IsSystem>1</IsSystem>
    <Description nlsid="SAMPLE_FACET_DESC">Files on a standard UNIX operating
system that contain configuration relevant to the networking operations.</
Description>
    <Author>SYSMAN</Author>
    <LastUpdatedBy>SYSMAN</LastUpdatedBy>
    <SourcePattern>
        <GeneralPattern>
            <Patterns>
                <Pattern>
                    <Value>{ETCDIR}/hosts</Value>
                    <Description nlsid="SAMPLE_FACET_PATTERN_1_DESC">Contains IP
to hostname mappings</Description>
                    <IsIncluded>1</IsIncluded>
                </Pattern>
                <Pattern>
                    <Value>{ETCDIR}/resolv.conf</Value>
                    <Description nlsid="SAMPLE_FACET_PATTERN_2_DESC">Contains
local name resolution mappings.</Description>
                    <IsIncluded>1</IsIncluded>
                </Pattern>
                <Pattern>
                    <Value>{ETCDIR}/appsecurity/*</Value>
                    <Description nlsid="SAMPLE_FACET_PATTERN_3_DESC">All files
in a directory used for my custom application.</Description>
                    <IsIncluded>1</IsIncluded>
```

```
                </Pattern>
                <Pattern>
                    <Value>{ETCDIR}/appsecurity/sample.conf</Value>
                    <Description nlsid="SAMPLE_FACET_PATTERN_4_DESC">Excluding
one file that is not a production configuration file that does not need to be
monitored.</Description>
                    <IsIncluded>0</IsIncluded>
                </Pattern>

            <Parameters>
                <Parameter>
                    <Name>ETCDIR</Name>
                    <Description nlsid="SAMPLE_FACET_PARAMETER_1_DESC">Location
where all base Unix configuration files sit.</Description>
                    <Value>/etc</Value>
                    <IsActive>1</IsActive>
                </Parameter>
            </Parameters>
        </GeneralPattern>
    </SourcePattern>
</Facet>
```

## Creating Real-time Monitoring Facets for Time Windows

Time windows are a special type of facet that is used for filtering real-time monitoring. Typically, the Enterprise Manager end user creates time window facets since they are specific to their own operations schedules, but this document includes the content for reference purposes.

Table 13-3 provides a description of the tags of a time window facet:

**Table 13-3    Key Tags Used to Define a Time Window Facet**

| Tag | Subtag | Description |
| --- | --- | --- |
| Name | NA | The internal name of the facet. This must be unique across all facets that exist and is not visible on the UI. |
| DisplayName | NA | The display name of the facet. It provides the nlsid attribute to support the translation of messages. |
| TargetType | NA | The type of target to which this rule is associated |
| EntityType | NA | The entity type for which you are creating the facet. For this example, it is timewindow |
| IsSystem | NA | True, for out-of-the-box rules. Otherwise, False. |
| Description | NA | The description of the facet. It provides the nlsid attribute to support the translation of messages. |
| Author | NA | The Enterprise Manager user that is the author of the facet. |

**Table 13-3    (Cont.) Key Tags Used to Define a Time Window Facet**

| Tag | Subtag | Description |
| --- | --- | --- |
| `LastUpdatedBy` | `NA` | The Enterprise Manager user that last updated the facet. This should be same as the author for your initially created data. |
| `SourcePattern/ SchedulePattern` | `NA` | NA |
| `NA` | `TZDisplayName` | The display name of the time zone in English. For example Greenwich Mean Time (UTC+0). |
| `NA` | `Duration` | • `DurStartMinute`: The minute starting from 00:00 (midnight) when the time window starts. For example, 1439 = 11:59PM<br>• `DurEndMinute`: The minute starting from 00:00 (midnight) when the time window ends<br>• `DurMinute`: Precalculated duration that can be used for describing the time window, especially if it spans two days. A duration must be less or equal to 1440 (24 hours) |
| `NA` | `Recurrence` | • `RecStartDate`: The date that the time window starts<br>• `RecurrencePattern`:<br><br>`RecPattern`: The type of recurrence. Options are: "Single", "Daily", "Weekly", "Monthly", or "Yearly".<br><br>`RecPatternDays`:Represents the days of the week, comma separated values. Sunday = 1, Saturday = 7.<br><br>`RecPatternFrequency`: The frequency for repeating if the type of recurrence is to do "Every X of some pattern". |

**Example: Sample Time Window Facet Definition**

```
<Facet is_time_window="1"
    <Name>general_working_hours</Name>
    <DisplayName>General Working Hours</DisplayName>
    <TargetType>host</TargetType>
    <EntityType>timewindow</EntityType>
    <IsSystem>0</IsSystem>
    <Description>Define the work hour from 9:00 am to 5:00 pm</Description>
    <Author>SYSMAN</Author>
    <LastUpdatedBy>SYSMAN</LastUpdatedBy>
    <SourcePattern>
        <SchedulePattern>
            <TZDisplayName/>
```

```
                    <Duration>
                        <DurStartMinute>540</DurStartMinute>
                        <DurEndMinute>1020</DurEndMinute>
                        <DurMinute>480</DurMinute>
                    </Duration>
                    <Recurrence>
                        <RecStartDate>2010-07-26</RecStartDate>
                        <RecurrencePattern>
                            <RecPattern>WEEKLY</RecPattern>
                            <RecPatternDays>1,2,5</RecPatternDays>
                            <RecPatternFrequency>1</RecPatternFrequency>
                        </RecurrencePattern>
                    </Recurrence>
                </SchedulePattern>
            </SourcePattern>
        </Facet>
```

## Creating Real-time Monitoring Rules

This section provides an overview of the XML tags used in creating a real-time monitoring rule and an example XML fragment showing rule creation. This XML fragment assumes that the facet has been created already and is referenced in this rule.

Table 13-4 provides a description of the tags used to define a real-time rule:

**Table 13-4    Key Tags Used to Define a Real-time Rule**

| Tag | Subtag | Description |
| --- | --- | --- |
| DisplayName | NA | The display name of the rule. It provides the nlsid attribute to support the translation of messages. |
| TargetType | NA | The type of target to which this rule is associated |
| IsSystem | NA | True, for out-of-the-box rules. Otherwise, False. |
| Description | NA | The description of the rule. It provides the nlsid attribute to support the translation of messages. |
| Impact | NA | Impact if the rule violates (when rule is noncompliant). It provides the nlsid attribute to support the translation of messages |
| ViolationContext List | NA | Violation context defines a violation to a rule uniquely. Violation context lists columns from <Source> Query, which will be visible as a part of the violation. Each column must mark as key or non-key.<br><br>The mandatory target_guid column from <Source> query is implicitly added to the violation context and should not be included in the violation context explicitly. |

**Table 13-4    (Cont.) Key Tags Used to Define a Real-time Rule**

| Tag | Subtag | Description |
| --- | --- | --- |
| NA | Column | Metric Column name. Uses Attributes name and type:<br><br>• `DisplayLabel`: Display name for column. It provides the nlsid attribute to support the translation of messages<br>• `IsHidden`: True, if this column should not be displayed as a part of a violation context. Otherwise, False.<br>• `IsKey`: True, if this column is a key. |
| CheckSource | NA | Defines the data source for Rule evaluation. |
| NA | RealTimeMonitoringLogicDefinition | Defines data source for Real-time Monitoring Rule |
| NA | EntityType | The type of monitoring performed (that is, `osfile`, `osprocess`, `osuser`, and so on). A full list is available in What Entity Types Can I Monitor?. |
| NA | Facets | The collection of facets to refer to in this rule. Some facets can be monitoring facets and some might be filtering facets.<br><br>• `Facet Reference`: The internal reference name of the facet<br><br>`Name`: The internal reference name of the facet that the rule refers to<br><br>`TargetType`: The target type of the referenced facet. This should always be the same as the rule target type for your content.<br><br>`EntityType`: The entity type of the referenced facet<br><br>`IsFilteredFacet`: **0** indicates this facet reference is used to determine what to monitor. **1** indicates this facet reference is a filter.<br><br>`InvertedFilteredFacet`: Only applicable if `IsFilteredFacet=1`. This specifies that the patterns in the facet definition are inverted (1) or not (0). If a filter facet was for "Production Hours" and then it was inverted, then monitoring will only occur outside of the pattern defined for "Production Hours" |
| NA | ObservationTypes/ObservationType | The types of observations you want to monitor in real-time.<br><br>`Name`: Internal reference name for the observation type you want to have monitored in this rule. |

**Table 13-4    (Cont.) Key Tags Used to Define a Real-time Rule**

| Tag | Subtag | Description |
|---|---|---|
| NA | Settings | • `CMSetting`: Settings related to the performance of change management reconciliation. |
| | | The auto_authorized=0 attribute indicates manual reconciliation only. 1 indicates integration with a change management server. Typically, you cannot use this field because the connector would not exist yet. If you create a rule without CM settings, a customer can override the rule and set their own custom CM settings after associating the rule with a Compliance Standard. |
| | | `CMConnector`: The connector the rule should use for automatic reconciliation. |
| | | `AnnotateAuthObservation`: Indicates whether the Change Management connector should annotate authorized observations into the requests that made the observations authorized. |
| NA | AdvancedSetting | Advanced rule settings |
| | | • `GroupSetting`: Settings about how observation bundles will be closed. Observation bundles collect a series of actions that happen against the same rule, by the same user, and on the same target over a short period of time. |
| | | `ObsGroupIdleTimeout`: The timeout period (in minutes) after the last user action before a bundle will be closed. |
| | | `ObsGroupMaxAge`: The maximum duration (in minutes) of an observation bundle. |
| | | `ObsGroupMaxObservations`: The maximum number of observations in an observation bundle. |
| | | • `GenerateEventByManualAuth`: If you are using manual reconciliation, then you have the option of generating informational level events when observations occur. 1 indicates that an informational event will be created for each observation group. 0 indicates that no informational event will be created. |
| NA | Options | Additional options that can be configured based on the entity type. Some entity types will not have options. |
| | | • `Option` (Name/Value): A single name or value pair option setting. |
| Severity | NA | Severity of the rule (Critical, Warning, or MinorWarning) |
| LifeCycleStatus | NA | Lifecycle status of the rule (Development or Production) |

**Table 13-4    (Cont.) Key Tags Used to Define a Real-time Rule**

| Tag | Subtag | Description |
|---|---|---|
| UrlLink | NA | Detail URL for the rule, containing details about the rule |
| ViolationMessage | NA | Message recorded with violation (for the rule). Used for notifications. It provides the nlsid attribute to support the translation of messages. |
| ClearViolationMe ssage | NA | Message recorded with clearing of violation (for the rule). Used for notifications. It provides the nlsid attribute to support the translation of messages. |
| KeywordList | NA | List of keywords associated with the rule |
| NA | Keyword | Keywords applicable to the compliance standard |
| Author | NA | Rule author |

**Example: Sample Rule Definition**

```
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
Name="monitor_critical_os_config_files">
    <DisplayName nlsid="SAMPLE_RULE_NAME">Monitor critical OS
configuration files</DisplayName>
    <TargetType>host</TargetType>
    <IsSystem>True</IsSystem>
        <Description nlsid="SAMPLE_RULE_DESC">Monitor several critical
        configuration areas of a Linux host to ensure no configuration
changes are
        happening out of bounds. Monitoring is only done during
production
        hours.</Description>
    <Impact nlsid="SAMPLE_RULE_IMPACT">Capturing real-time changes to
these files may indicate a serious security issue.</Impact>
    <Recommendation nlsid="SAMPLE_RULE_RECO">Ensure that change
management policy documents how and when changes should be made in
production.
        Create compensating controls to address these out of bound
issues.</Recommendation>
    <ViolationContextList/>
    <CheckSource>
        <RealTimeMonitoringLogicDefinition>
            <EntityType>osfile</EntityType>
            <Facets>
                <FacetReference>
                    <Name>network_configuration_files</Name>
                    <TargetType>host</TargetType>
                    <EntityType>osfile</EntityType>
                    <IsFilteredFacet>0</IsFilteredFacet>
                    <InvertFilteredFacet>0</InvertFilteredFacet>
                </FacetReference>
                <FacetReference>
                    <Name>maild_configuration_files</Name>
                    <TargetType>host</TargetType>
```

```xml
            <EntityType>osfile</EntityType>
            <IsFilteredFacet>0</IsFilteredFacet>
            <InvertFilteredFacet>0</InvertFilteredFacet>
        </FacetReference>
        <FacetReference>
            <Name>sshd_configuration_files</Name>
            <TargetType>host</TargetType>
            <EntityType>osfile</EntityType>
            <IsFilteredFacet>0</IsFilteredFacet>
            <InvertFilteredFacet>0</InvertFilteredFacet>
        </FacetReference>
        <FacetReference>
            <Name>crontab_configuration_files</Name>
            <TargetType>host</TargetType>
            <EntityType>osfile</EntityType>
            <IsFilteredFacet>0</IsFilteredFacet>
            <InvertFilteredFacet>0</InvertFilteredFacet>
        </FacetReference>
        <FacetReference>
            <Name>kernel_configuration_files</Name>
            <TargetType>host</TargetType>
            <EntityType>osfile</EntityType>
            <IsFilteredFacet>0</IsFilteredFacet>
            <InvertFilteredFacet>0</InvertFilteredFacet>
        </FacetReference>
        <FacetReference>
            <Name>production_hours</Name>
            <TargetType>host</TargetType>
            <EntityType>timewindow</EntityType>
            <IsFilteredFacet>1</IsFilteredFacet>
            <InvertFilteredFacet>0</InvertFilteredFacet>
        </FacetReference>
    </Facets>
    <ObservationTypes>
        <ObservationType>
            <Name>osfile_create_suc</Name>
        </ObservationType>
        <ObservationType>
            <Name>osfile_content_modified_suc</Name>
        </ObservationType>
        <ObservationType>
            <Name>osfile_delete_suc</Name>
        </ObservationType>
        <ObservationType>
            <Name>osfile_content_mod_archive_suc</Name>
        </ObservationType>
    </ObservationTypes>
    <Settings>
        <CMSetting auto_authorized="0">
            <CMConnector></CMConnector>
            <AnnotateAuthObservation></AnnotateAuthObservation>
        </CMSetting>
        <AdvancedSetting>
            <GroupSetting>
                <ObsGroupIdleTimeout>15</ObsGroupIdleTimeout>
```

```
                             <ObsGroupMaxAge>30</ObsGroupMaxAge>
                             <ObsGroupMaxObservations>1000</
ObsGroupMaxObservations>
                        </GroupSetting>
                        <GenerateEventByManualAuth>0</
GenerateEventByManualAuth>
                    </AdvancedSetting>
                </Settings>
                <Options>
                    <Option value="10" name="osfile_archivenumber"/>
                    <Option value="50000"
name="osfile_polling_maxfilealert"/>
                    <Option value="100"
name="osfile_archive_maxsrcfilealert"/>
                </Options>
        </RealTimeMonitoringLogicDefinition>
    </CheckSource>
    <Severity>MinorWarning</Severity>
    <LifeCycleStatus>Development</LifeCycleStatus>
    <KeywordList>
        <Keyword nlsid="CONFIGURATION">Configuration</keyword>
        <Keyword nlsid="SECURITY">Security</keyword>
    </KeywordList>
    <ViolationMessage nlsid="SAMPLE_RULE_VIOL_MSG">Violation due to
change in critical OS configuration files during production hours.</
ViolationMessage>
    <ClearViolationMessage nlsid="SAMPLE_RULE_VIOL_CLRMSG">Cleared
violation due to change in critical OS configuration files during
production hours.</ClearViolationMessage>
    <Author>SYSMAN</Author>
</Rule>
```

# Defining Compliance Standards

Compliance Standards are mapped to Compliance Standard Rules (Repository Rules or Real-time Monitoring Rules) in a hierarchical fashion.

The following example provides the syntax for defining compliance standards and the next example provides an example of a Compliance Standard Definition.

> **✎ Note:**
>
> For the complete compliance XSDs, see the following JAR file:
>
> `$ORACLE_HOME/sysman/jlib/gccomplianceCommon.jar`

> **✎ Note:**
>
> For additional examples, see More Compliance Examples.

**Example: Compliance Standard Definition Syntax**

```
<xsd:complexType name="StandardT">
    <xsd:sequence>
      <xsd:element name="DisplayName" type="std:DisplayString128Def"
minOccurs="0"/>
      <xsd:element name="TargetType" type="std:Name128Def"
minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="std:TargetPropertyFilter" minOccurs="0"/>
      <xsd:element name="Author" type="std:Name256Def" default="ORACLE"
minOccurs="0"/>
      <xsd:element name="Version" type="xsd:nonNegativeInteger"
default="1" minOccurs="0"/>
      <xsd:element name="LifeCycleStatus" default="Development"
minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Development"/>
            <xsd:enumeration value="Production"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="IsHidden" type="std:BooleanDef" minOccurs="0"
default="false"/>
      <xsd:element name="IsSystem" type="std:BooleanDef" minOccurs="0"
default="false"/>
      <xsd:element name="IsAutoEnable" type="std:BooleanDef"
minOccurs="0" default="false"/>
      <xsd:element name="Description" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="KeywordList" type="std:KeywordListT"
minOccurs="0"/>
      <xsd:element name="ReferenceURL" type="std:String4000Def"
minOccurs="0"/>
      <xsd:element name="FrontMatter" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="RearMatter" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="Notice" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="Body" type="std:BodyT" minOccurs="0"/>
      <xsd:element name="ExtraInfo" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="std:NameDef" use="required"/>
    <xsd:attribute name="oms_version" type="std:Name32Def"
use="required"/>
  </xsd:complexType>
```

Table 13-5 provides a description of the tags used in defining Compliance Standards:

**Table 13-5    Key Tags Used in Defining Compliance Standards**

| Tag | Subtag | Description |
| --- | --- | --- |
| DisplayName | NA | The display name of the compliance standard. It provides the nlsid attribute to support the translation of messages.<br><br>**Note**: The nlsid attribute is not applicable to metadata plug-ins. |
| TargetType | NA | The type of target to which this compliance standard can be associated |
| Author | NA | Compliance standard author |
| Version | NA | The version of the compliance standard |
| LifeCycleStatus | NA | Lifecycle status of compliance standard (Development or Production) |
| IsSystem | NA | True, if the compliance standard is provided out-of-the-box. Otherwise, False. |
| Description | NA | Description of the compliance standard. It provides the nlsid attribute to support the translation of messages. |
| IsAutoEnable | NA | If set to True, the compliance standard will be associated with all exiting targets for the defined target type. (Defined using TargetType) |
| KeywordList | NA | A list of keywords applicable to the compliance standard |
| NA | Keyword | Keywords applicable to the compliance standard |
| ReferenceURL | NA | The reference URL of the compliance standard |
| FrontMatter | NA | Front matter message. It provides the nlsid attribute to support the translation of messages. |
| RearMatter | NA | Rear matter message. It provides the nlsid attribute to support the translation of messages. |
| Notice | NA | Notice message. It provides the nlsid attribute to support the translation of messages. |
| Body | NA | Body of the compliance standard. Can have one or more of the following listed elements |

**Table 13-5    (Cont.) Key Tags Used in Defining Compliance Standards**

| Tag | Subtag | Description |
| --- | --- | --- |
| NA | RuleFolder | Defines a rule folder. A RuleFolder can have the following:<br><br>**RuleFolder**<br><br>**RuleReference**<br><br>**Include Standard Reference**<br><br>• DisplayName: The display name of the Rule Folder. It provides the nlsid attribute to support the translation of messages.<br><br>• Description: Description of the Rule Folder. It provides the nlsid attribute to support the translation of messages.<br><br>**Note**: The nlsid attribute is not applicable to metadata plug-ins.<br><br>• ReferenceURL: The reference URL of the Rule Folder<br><br>• Importance: Importance of Rule Folder (Low/Normal/High) |
| NA | Include | Include another compliance standard reference to the including compliance standard |
| NA | RuleReference | Include rule reference to the compliance standard |

**Example: Sample Compliance Standard 1**

```
<Standard xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_cs1">
     <DisplayName nlsid="SAMPLE_CS_1_NAME">Sample Compliance Standard
1</DisplayName>
    <TargetType>oracle_database</TargetType>
                    <TargetPropertyFilter>
                                <PropertyItem>

<PropertyName>orcl_gtp_target_version</PropertyName>
                                            <ValueList>
                                                    <Value>8.1.6+</
Value>
                                            </ValueList>
                                </PropertyItem>
                        </TargetPropertyFilter>
    <Author>SYSTEM</Author>
    <Version>1</Version>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <IsSystem>true</IsSystem>
    <Description nlsid="SAMPLE_CS_1_DESC">Sample Description</
Description>
    <KeywordList>
        <Keyword nlsid="CONFIGURATION">Configuration</Keyword>
    </KeywordList>
    <ReferenceURL>http://sampleurl.com</ReferenceURL>
    <Body>
```

```
                            <RuleFolder name="sample_RF_1">
                                    <DisplayName
nlsid="SAMPLE_RF_1_NAME">Sample Rulefolder</DisplayName>
                                    <Description
nlsid="SAMPLE_RF_1_DESC">This includes rules that checks for use of a
single control file</Description>
                                    <ReferenceURL>http://
www.oracle.com/db_rf1</ReferenceURL>
                                    <Importance>Normal</Importance>
                                    <RuleReference>
                                    <Name>sample_rule1</Name>
                                    <TargetType>oracle_database</
TargetType>
                                    <Importance>Normal</Importance>
                                    </RuleReference>
                            </RuleFolder>
        </Body>
</Standard>
```

# Defining a Compliance Framework

> **Note:**
>
> Although the Compliance Framework term is used throughout this document, the XML API uses the term `Group` or `SubGroup`. This is an internal name used for the XML structure that is not exposed on the Enterprise Manager UI.

The following example provides the syntax for defining a compliance framework and the next example provides an example of a compliance framework definition.

> **Note:**
>
> For the complete compliance XSDs, see the following JAR file:
>
> `$ORACLE_HOME/sysman/jlib/gccomplianceCommon.jar`

> **Note:**
>
> For additional examples, see More Compliance Examples.

**Example: Compliance Framework Definition Syntax**

```
<xsd:complexType name="StandardGroupT">
    <xsd:sequence>
```

```
      <xsd:element name="DisplayName" type="std:DisplayString128Def"
minOccurs="0"/>
      <xsd:element name="Author" type="std:Name256Def" default="ORACLE"
minOccurs="0"/>
      <xsd:element name="Version" type="xsd:nonNegativeInteger"
default="1" minOccurs="0"/>
      <xsd:element name="LifeCycleStatus"
default="Development"minOccurs="0">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Development"/>
            <xsd:enumeration value="Production"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="Description" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="KeywordList" type="std:KeywordListT"
minOccurs="0"/>
      <xsd:element name="ReferenceURL" type="std:String4000Def"
minOccurs="0"/>
      <xsd:element name="FrontMatter" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="RearMatter" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="Notice" type="std:DisplayString800Def"
minOccurs="0"/>
      <xsd:element name="IsHidden" type="std:BooleanDef"
minOccurs="0"default="false"/>
      <xsd:element name="IsSystem" type="std:BooleanDef"
minOccurs="0"default="false"/>
      <xsd:element name="GroupBody" type="std:GroupBodyT"
minOccurs="0"/>
      <xsd:element name="ExtraInfo" type="xsd:string"
minOccurs="0"maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="std:NameDef" use="required"/>
    <xsd:attribute name="oms_version" type="std:Name32Def"
use="required"/>
  </xsd:complexType>
```

Table 13-6 provides a description of the tags used in defining a Compliance Framework:

**Table 13-6    Key Tags Used in Defining a Compliance Framework**

| Tag | Subtag | Description |
| --- | --- | --- |
| DisplayName | NA | The display name of the compliance framework. It provides the nlsid attribute to support the translation of messages. |
| Author | NA | Author of the compliance framework |
| Version | NA | The version of the compliance framework |

**Table 13-6    (Cont.) Key Tags Used in Defining a Compliance Framework**

| Tag | Subtag | Description |
| --- | --- | --- |
| LifeCycleStatus | NA | The lifecycle status of the compliance framework (Development or Production) |
| IsSystem | NA | True, if compliance framework is provided out-of-the box. Otherwise, False. |
| Description | NA | Description of compliance framework. It provides the nlsid attribute to support the translation of messages. **Note**: The nlsid attribute is not applicable to metadata plug-ins. |
| KeywordList | NA | List of keywords applicable to compliance framework |
| NA | Keyword | Keywords applicable to the compliance standard |
| ReferenceURL | NA | The reference URL of the compliance framework |
| FrontMatter | NA | Front matter message. It provides the nlsid attribute to support the translation of messages |
| RearMatter | NA | Rear matter message. It provides the nlsid attribute to support the translation of messages. |
| Notice | NA | Notice message. It provides the nlsid attribute to support the translation of messages. |
| ExtraInfo | NA | Additional information about the compliance framework. |
| GroupBody | NA | Defines the body of the compliance framework. It can have one or more of the following elements: |
| NA | SubGroup | Defines a child framework element. A child framework element can include the following: **Child framework** **Include Standard Reference**. <ul><li>DisplayName: The display name of the child framework</li><li>Description: Description of the child framework</li><li>ReferenceURL: The reference URL of the child framework</li><li>Importance: Importance of child framework (Low, Normal, or High)</li></ul> |
| NA | StandardReference | Includes the compliance standard reference to the compliance framework |

**Example: Sample Compliance Framework**

```
<StandardGroup xmlns="http://www.oracle.com/DataCenter/ConfigStd"
name="sample_csg" oms_version="12.1.0.1.0">
        <DisplayName nlsid="SAMPLE_CSG_NAME">Sample Compliance
```

```
Framework</DisplayName>
        <Author>SYSTEM</Author>
        <Version>1</Version>
        <LifeCycleStatus>Production</LifeCycleStatus>
        <Description nlsid="SAMPLE_CSG_DESC">Sample Description</
Description>
        <KeywordList>
            <Keyword nlsid="SECURITY">Security</Keyword>
        </KeywordList>
        <ReferenceURL>http://sampleurl.com</ReferenceURL>
        <IsHidden>false</IsHidden>
        <IsSystem>true</IsSystem>
        <GroupBody>
            <SubGroup name="SampleSubgroup">
                <DisplayName nlsid="SAMPLE_CSG_SUBGROUP_NAME">Sample
Child Framework</DisplayName>
                <Description nlsid="SAMPLE_CSG_SUBGROUP_DESC">Sample
Child framework Description</Description>
                <ReferenceURL>http://sampleurl.com</ReferenceURL>
                <Importance>Normal</Importance>
                <StandardReference>
                    <Name>sample_cs3</Name>
                    <Author>SYSTEM</Author>
                    <Version>1</Version>
                    <Importance>Normal</Importance>
                </StandardReference>
            </SubGroup>
        </GroupBody>
 </StandardGroup>
```

# Defining Compliance Content

The following example provides the syntax for defining compliance content and the
Sample XML Compliance Metadata example provides an example of XML compliance
metadata.

> **Note:**
>
> For additional examples, see More Compliance Examples.

**Example: Compliance Content Definition Syntax**

```
<xsd:complexType name="ComplianceContentT">
  <xsd:sequence>

    <!-- Cummulative change since the first release.-->
    <xsd:element ref="std:ChangeList" minOccurs="0" maxOccurs="1"/>
    <!-- End Cummulative change since the first release -->

    <!-- Current state of entities -->
    <xsd:element ref="std:Facet" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="std:Rule" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="std:Standard" minOccurs="0" maxOccurs="unbounded"/>
```

```
     <xsd:element ref="std:StandardGroup" minOccurs="0" maxOccurs="unbounded"/>
     <!-- Current state of entities -->
   </xsd:sequence>
   <xsd:attribute name="oms_version" type="std:Name32Def" use="required"/>
   <xsd:attribute name="name" type="std:Name64Def" use="required"/>
   <!-- content_version of compliance content should be equal to version of last
change tag if any. -->
   <xsd:attribute name="content_version" type="std:Name64Def" use="optional"
default = "12.1.0.0.0"/>
   <xsd:attribute name="IsCompareEnabled" type="std:BooleanDef" use="optional"
default = "true"/>
</xsd:complexType>

<xsd:element name="ComplianceContent" type="std:ComplianceContentT"/>
```

Table 13-7 provides a description of some of the attributes used in defining compliance content:

**Table 13-7    Compliance Content Attributes**

| Attribute | Description |
| --- | --- |
| oms_version | Version of Oracle Management Service (OMS) |
| name | Name of the compliance content |
| content_version | Version of the compliance content |
| IsCompareEnabled | Specifies whether a rule or compliance standard is updated incrementally or if the entire rule or compliance standard is regenerated. |
| | **Possible Values**: |
| | • True: For each rule and standard tag, the software finds the incremental change automatically and updates the entity incrementally. For example, if only one rule is updated in a compliance standard, only that rule is updated in the compliance standard and then the updated rule is reevaluated for all targets associated to the compliance standard at the time of the rule update (where the rule is a repository rule) |
| | • False: The user must specify `<UpdateRule>` within the `<ChangeList><Change..>....</ChangeList></Change>` tags. This causes the rule to be overridden ( that is, all attributes and definitions). |
| | Similarly, if a compliance standard is updated, it will override the standard completely and and regenerate results (in case of repository check-based standards). |
| | **Note**: If you set `isCompareEnabled = false`, then you must provide all the changes that were made in each version cumulatively since the compliance content was created. This is very important for metadata consistency. |
| | Oracle recommends that you always summarize the changes in each version even if the `isCompareEnabled` attribute is set to true. Because if you need to switch from `isCompareEnabled= true` (default) to `isCompareEnabled=false` at a future date, then all historical changes across different versions of the compliance content will be available to you. |

**Example: Sample XML Compliance Metadata**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE ComplianceContent [
<!ENTITY rule SYSTEM "SampleRuleThresholdCondition.xml">
<!ENTITY standard SYSTEM "SampleComplianceStandard.xml">
<!ENTITY standardgroup SYSTEM "SampleComplianceStandardGroup.xml">
]>
<ComplianceContent xmlns="http://www.oracle.com/DataCenter/ConfigStd"
name="Sample Compliance Framework" oms_version="11.2.0.1.0">
<ChangeList>
                <Change version="12.2.0.0.0">
                        <UpdateRule>
                                <RuleName>sample_rule</RuleName>
                                <TargetType>oracle_database</TargetType>
                        </UpdateRule>
                        <UpdateStandardGroup>
                                <StandardGroupName>sample_csg</
StandardGroupName>

                                <StandardGroupAuthor>SYSTEM</
StandardGroupAuthor>

                                <StandardGroupVersion>1</
StandardGroupVersion>
                        </UpdateStandardGroup>
                </Change>
</ChangeList>
&rule;
&standard;
&standardgroup;
</ComplianceContent>
```

# Removing Compliance Content

To remove or delete compliance content, enter the following command:

```
emctl deregister oms metadata -sysman_pwd sysman -core -service gccompliance
-file DeleteComplianceContent.xml
```

In the previous command, *DeleteComplianceContent.xml* represents the name of the Delete Compliance Content XML file.

The following examples provide the syntax for defining Delete Compliance Content and an example of a Delete Compliance Content XML file.

### Example: Delete Compliance Content Syntax

```
<!-- delete compliance metadata corresponding to the compliance content name
provided. -->
  <xsd:complexType name="DeleteComplianceContentT">
    <xsd:attribute name="name" type="std:Name64Def" use="required"/>
  </xsd:complexType>
<xsd:element name="DeleteComplianceContent" type="std:DeleteComplianceContentT"/>
```

### Example: DeleteComplianceContent XML

```
  <DeleteComplianceContent xmlns="http://www.oracle.com/DataCenter/ConfigStd"
name="Sample Compliance Framework" />
```

# Supporting Translation

> **✎ Note:**
>
> Translation is supported for the Oracle Fusion Middleware plug-in only.

For each `nlsid` attribute in the XML samples, you must specify a Data Loading Format (DLF) map entry. A DLF file contains the English string for each defined `nlsid` attribute. These strings are available for translation.

**Example: Sample DLF File**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<table xml:lang="en" name="MGMT_MESSAGES">

<!-- lookup-key indicates which columns are used by
     TransX to recognize a row as a duplicate -->
<lookup-key>
  <column name="MESSAGE_ID"/>
  <column name="SUBSYSTEM"/>
  <column name="LANGUAGE_CODE"/>
  <column name="COUNTRY_CODE"/>
</lookup-key>

<!-- columns field indicates which columns will be loaded as
     part of processing the dataset and which should be
     translated by the Translation Group -->
<columns>
  <column name="MESSAGE_ID" type="string" maxsize="64"/>
  <column name="SUBSYSTEM" type="string" maxsize="64"/>
  <column name="LANGUAGE_CODE" type="string" language="%l"/>
  <column name="COUNTRY_CODE" type="string" language="%Cs"/>
  <column name="MESSAGE" type="string" maxsize="1000" translate="yes"/>
</columns>

<!-- dataset specifies the data to be loaded into the repository -->
<dataset>

  <row>
    <col name="MESSAGE_ID">SAMPLE_RULE_NAME</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Rule</col>
  </row>

  <row>
    <col name="MESSAGE_ID">SAMPLE_RULE_DESC</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Checks for use of a single control file</col>
  </row>

 <row>
    <col name="MESSAGE_ID">SAMPLE_RULE_IMPACT</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">The control file is one of the most important files in an
Oracle database. It maintains many physical characteristics and important
```

```
recovery information about the database. If you lose the only copy of the control
file due to a media error, there will be unnecessary down time and other risks.</
col>
 </row>

<row>
    <col name="MESSAGE_ID">SAMPLE_RULE_RECO</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Use at least two control files that are multiplexed on
different disks.</col>
 </row>

<row>
    <col name="MESSAGE_ID">SAMPLE_RULE_COL_1</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">FILE_LIST</col>
 </row>

 <row>
    <col name="MESSAGE_ID">SAMPLE_RULE_COL_2</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">CONTROL_FILE_COUNT</col>
 </row>

<row>
    <col name="MESSAGE_ID">SAMPLE_RULE_VIOL_MSG</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">The database has an insufficient number of control
files.</col>
 </row>

<row>
    <col name="MESSAGE_ID">SAMPLE_RULE_VIOL_CLEAR_MSG</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">The database has sufficient number of control files.</
col>
</row>

<!-- Standard NLSID Mappings -->

<row>
    <col name="MESSAGE_ID">SAMPLE_CS_NAME</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Compliance Standard</col>
 </row>

 <row>
    <col name="MESSAGE_ID">SAMPLE_CS_DESC</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Description</col>
 </row>

<row>
    <col name="MESSAGE_ID">SAMPLE_RF_NAME</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Rulefolder</col>
</row>

 <row>
    <col name="MESSAGE_ID">SAMPLE_RF_DESC</col>
    <col name="SUBSYSTEM">POLICY</col>
```

```
      <col name="MESSAGE">This includes rules that checks for use of a single
control file.</col>
 </row>

<!-- Standard Group NLSID Mappings -->

 <row>
    <col name="MESSAGE_ID">SAMPLE_CSG_NAME</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Compliance Framework</col>
 </row>

 <row>
    <col name="MESSAGE_ID">SAMPLE_CSG_DESC</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Description</col>
 </row>

 <row>
    <col name="MESSAGE_ID">SAMPLE_CSG_SUBGROUP_NAME</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Child Framework</col>
 </row>

  <row>
    <col name="MESSAGE_ID">SAMPLE_CSG_SUBGROUP_DESC</col>
    <col name="SUBSYSTEM">POLICY</col>
    <col name="MESSAGE">Sample Child Framework Description</col>
 </row>

</dataset>
</table>
```

> ✎ **Note:**
>
> If the DLF entry is for a real-time monitoring facet or pattern, then the subsystem is GCCOMPLIANCE_CCC. For all other rules, the subsystem is POLICY.

# Packaging Compliance XML

This section indicates the location of the XML and DLF files.

- XML Files

  Store all the XML files in the following directory:

  *plugin_stage*/oms/metadata/gccompliance/

  In the previous directory path, *plugin_stage* is the plug-in staging directory.

  For more information about the plug-in staging directory, see Staging the Plug-in.

- DLF Files

  Store all the DLF files in the following directory:

  *plugin_stage*/oms/rsc/*area*/gccompliance

In the previous directory path, *plugin_stage* is the plug-in staging directory and *area* represents the subcomponent such as db for database or ecm for configuration management.

# Setting Up and Testing Compliance Standards and Rules

To test your compliance standards or rules, do the following:

- Install Compliance Content
- Test Compliance Standard

## Install Compliance Content

To install compliance content:

1. Use the following command to install the compliance content:

   ```
   emctl register oms metadata -sysman_pwd password -core -service gccompliance
   -file ComplianceContent.xml
   ```

2. Submit the following job for back-end processing:

   ```
   begin em_compliance_util.trigger_rule_dependency_job;end;
   ```

## Test Compliance Standard

To test your compliance standard:

1. Log in to the Cloud Control console.

2. From the **Enterprise** menu, select **Compliance**, then select **Library**.

   The Compliance Library page appears.

3. Click **Compliance Standards**.

4. Select the required compliance standard, then click **Associate Targets**.

   The Target Association for Compliance Standard: *Compliance Standard Name* page appears, where *Compliance Standard Name* is the name of your selected compliance standard.

5. Click **Add**.

   The Search and Select: Targets window appears.

6. Select the target that you want to evaluate, then click **Select**.

7. From the Target Association for Compliance Standard: *Compliance Standard Name* page, click **OK**.

8. Click **Yes** to the Save Association message.

   The Compliance Standards page appears.

   The previous steps trigger the evaluation, which occurs in a background job.

9. After a few minutes, from the **Enterprise** menu, select **Compliance**, then select **Results**.

   The Compliance Standards Evaluation Results page appears.

10. Select your compliance standard, then click **Show Details**.

The Compliance Standard Result Detail page appears.

11. From the left-hand side of the page, expand *Compliance Standard Name* to view any nodes, then click a node to view the results for that node.

**Figure 13-1    Compliance Standard Result Detail**
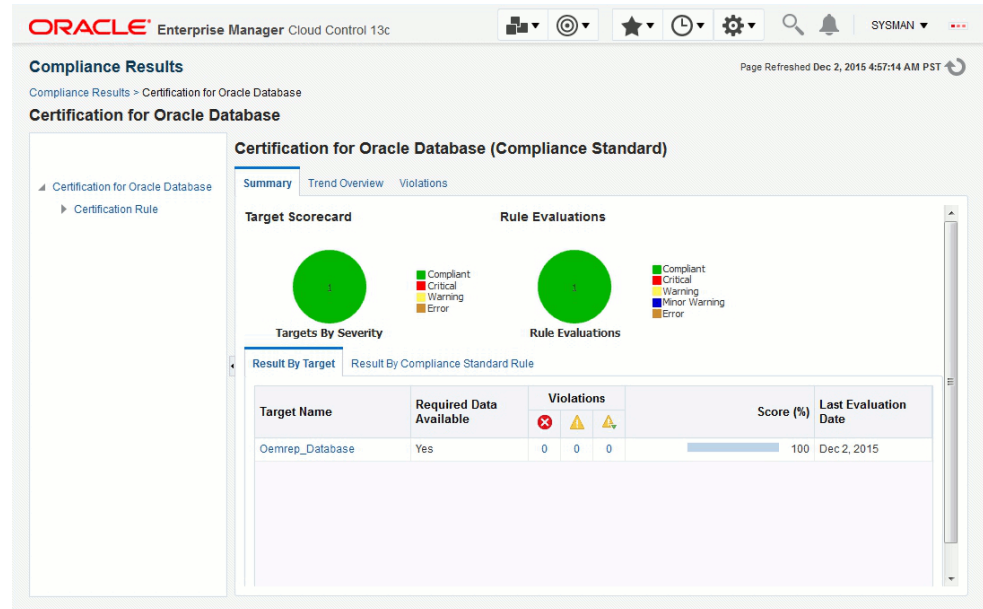


## Constraints for Testing

Note the following constraints when you are testing your compliance standards or rules:

- The MGMT_VIEW user must have the SELECT privilege on the views used in the query

- target_guid must be one of the SELECT attributes in the query

- Alias names or select clause names must be less than 64 characters

- Ensure that the standard references from a compliance standard are imported first. Place the standard references first in the compliance content list.

- At least one column from the SELECT clause of the SQL source must be marked as a non-key column in the violation context definition and metric definition.

- The target_guid column must *not* be specified for violation context columns or for metric definitions.

- If the query references views from outside of the enclosing plug-in, then the views must be exposed by the EDK to the plug-in (at the plug-in EDK level).

- If the SQL source query of a repository rule refers to a PLSQL function, then ensure that it refers to global PLSQL functions only, and not package functions (that is, if those PLSQL functions depend on tables whose update triggers a rule evaluation). This is required to generate the list of tables which the rule evaluation outcome depends on correctly. Execute privileges must be granted to the mgmt_view user on this function.

- The target type of the rule included in a compliance standard must be the same as that of the immediate parent standard.
- Key columns of STRING type must contain less than 64 characters.

# More Compliance Examples

This section provides additional examples of compliance content, rules, compliance standards, and compliance framework.

> **Note:**
>
> The examples in this chapter apply to both Enterprise Manager 12*c* and Enterprise Manager 13*c*.

The following example provides an example of compliance content version 1 and the next example provides an example of compliance content version 2. Version 1 is the initial version of the compliance content. Note that the content version number is 12.1.0.1.0, while the content version in the Compliance Content Version 2 example is 12.1.0.2.0.

Compliance content contains a `ChangeList` element. The `ChangeList` element describes the changes that have occurred since the first version of compliance content, such as updated rules, standards, and so on.

**Example: Compliance Content Version 1**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE ComplianceContent [
<!ENTITY rule1 SYSTEM "SampleRule1.xml">
<!ENTITY rule2 SYSTEM "SampleRule2.xml">
<!ENTITY rule3 SYSTEM "SampleRule3.xml">
<!ENTITY rule4 SYSTEM "SampleRule4.xml">
<!ENTITY rule5 SYSTEM "SampleRule5.xml">
<!ENTITY rule6 SYSTEM "SampleRule6.xml">
<!ENTITY standard1 SYSTEM "SampleComplianceStandard1.xml">
<!ENTITY standard2 SYSTEM "SampleComplianceStandard2.xml">
<!ENTITY standard3 SYSTEM "SampleComplianceStandard3.xml">
<!ENTITY standardgroup SYSTEM "SampleComplianceFramework.xml">]
]>
<ComplianceContent xmlns="http://www.oracle.com/DataCenter/ConfigStd"
name="SampleComplianceContent" oms_version="12.1.0.1.0"
content_version="12.1.0.1.0">
<ChangeList>
    <!-- ChangeList tag process each of the Change Tag with respect to the
version of the ComplianceContent installed in repository. -->
            <Change version="12.1.0.1.0">
        <!-- AddSubGroupWithinStandardGroup will introduce a subgroup within an
existing  compliance framework/standard group in repository. -->
        <!-- AddStandardReferenceToStandardGroup will introduce a reference
to a standard within an existing  compliance framework/standard group in
repository. -->
                <AddSubGroupWithinStandardGroup order="2">
                        <StandardGroupName>oracle_pci</StandardGroupName>
                        <StandardGroupAuthor>ORACLE</StandardGroupAuthor>
                        <StandardGroupVersion>1</StandardGroupVersion>
```

```
                                <SubGroup name="sampleSubgroup1">
                                        <DisplayName>sub1</DisplayName>
                                        <ReferenceURL>http://
sampleAddedSubgroup.com</ReferenceURL>
                                        <Importance>High</Importance>
                                </SubGroup>
            </AddSubGroupWithinStandardGroup>
                        <AddStandardReferenceToStandardGroup>
                                <StandardGroupName>oracle_pci</StandardGroupName>
                                <StandardGroupAuthor>ORACLE</StandardGroupAuthor>
                                <StandardGroupVersion>1</StandardGroupVersion>
                                <SubGroupListInfo>
                                        <SubGroupElem>oracle_pci_ctrlobj_a</
SubGroupElem>
                                </SubGroupListInfo>
                                <StandardReference>
                                        <Name>sample_cs1</Name>
                                        <Author>SYSTEM</Author>
                                        <Version>1</Version>
                                </StandardReference>
        </AddStandardReferenceToStandardGroup>
            </Change>
</ChangeList>
<!--List of compliance standard rules -->
&rule1;
&rule2;
&rule3;
&rule4;
&rule5;
&rule6;
<!--List of compliance standards -->
&standard1;
&standard2;
&standard3;
<!--List of compliance standard groups/frameworks-->
&standardgroup;
</ComplianceContent>
```

The following example provides an example of a compliance rule that checks for use of a single control file

**Example: Sample Rule 1**

```
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_rule1">
    <DisplayName nlsid="SAMPLE_RULE_1_NAME">Sample Rule 1</DisplayName>
    <TargetType>oracle_database</TargetType>
    <IsSystem>true</IsSystem>
    <TargetPropertyFilter>
        <PropertyItem>
            <PropertyName>orcl_gtp_operating_system</PropertyName>
            <ValueList>
                <Value>Windows</Value>
            </ValueList>
        </PropertyItem>
        <PropertyItem>
            <PropertyName>orcl_gtp_target_version</PropertyName>
            <ValueList>
                <Value>8.1.6+</Value>
            </ValueList>
        </PropertyItem>
```

```
            </TargetPropertyFilter>
        <Description nlsid="SAMPLE_RULE_1_DESC">Checks for use of a single control
file</Description>
        <Impact nlsid="SAMPLE_RULE_1_IMPACT">The control file is one of the most
important files in an Oracle database. It maintains many physical characteristics
and important recovery information about the database. If you lose the only copy
of the control file due to a media error, there will be unnecessary down time and
other risks.</Impact>
        <Recommendation nlsid="SAMPLE_RULE_1_RECO">Use at least two control files
that are multiplexed on different disks.</Recommendation>
        <ViolationContextList>
            <Column type="String" name="FILE_LIST">
                <DisplayLabel nlsid="SAMPLE_RULE_1_COL_1">FILE_LIST</DisplayLabel>
                <IsHidden>false</IsHidden>
                <IsKey>false</IsKey>
            </Column>
            <Column type="Number" name="CONTROL_FILE_COUNT">
                <DisplayLabel nlsid="SAMPLE_RULE_1_COL_2">CONTROL_FILE_COUNT</
DisplayLabel>
                <IsHidden>false</IsHidden>
                <IsKey>false</IsKey>
            </Column>
        </ViolationContextList>
        <CheckSource>
            <RepositoryCheckDefinition>
                <Metric>
        <TargetType>oracle_database</TargetType>
        <MetricName>sample_rule1</MetricName>
        <SourceType>SQL</SourceType>
            <Source>select CONTROL_FILE_COUNT, FILE_LIST, TARGET_GUID from
MGMT$CS_DB_CONTROL_FILE_COUNT</Source>
        <MetricColumnList>
            <MetricColumnInfo>
                    <ColumnName>FILE_LIST</ColumnName>
                    <ColumnType>String</ColumnType>
                    <isKey>false</isKey>
                    <ColumnLabel nlsid="SAMPLE_RULE_1_COL_1">FILE_LIST</ColumnLabel>
                        </MetricColumnInfo>
            <MetricColumnInfo>
                    <ColumnName>CONTROL_FILE_COUNT</ColumnName>
                    <ColumnType>Number</ColumnType>
                    <isKey>false</isKey>
                    <ColumnLabel nlsid="SAMPLE_RULE_1_COL_2">CONTROL_FILE_COUNT</
ColumnLabel>
            </MetricColumnInfo>
        </MetricColumnList>
                </Metric>
            <ParameterList>
                <RuleParameter>
                    <ParamName>CONTROL_FILE_COUNT</ParamName>
                    <ParamType>Number</ParamType>
                </RuleParameter>
            </ParameterList>
            <ParameterDefaultSettings>
                <ParamValue>
                    <ParamName>CONTROL_FILE_COUNT</ParamName>
                    <MinorWarnThreshold>1</MinorWarnThreshold>
                </ParamValue>
            </ParameterDefaultSettings>
            <TestCondition>
                <ThresholdCriteria>
```

```
                              <ColumnName>CONTROL_FILE_COUNT</ColumnName>
                              <TestOperator>EQ</TestOperator>
                              <ThresholdValue>1</ThresholdValue>
                              <ThresholdType>Number</ThresholdType>
                        </ThresholdCriteria>
                  </TestCondition>
            </RepositoryCheckDefinition>
      </CheckSource>
    <Severity>MinorWarning</Severity>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <KeywordList>
        <Keyword nlsid="CONFIGURATION">Configuration</Keyword>
    </KeywordList>
    <ViolationMessage nlsid="SAMPLE_RULE_1_VIOL_MSG">The database has an
insufficient number of control files.</ViolationMessage>
    <ClearViolationMessage nlsid="SAMPLE_RULE_1_VIOL_CLEAR_MSG">The database has
sufficient number of control files.</ClearViolationMessage>
    <Author>SYSMAN</Author>
</Rule>
```

The following example provides a sample compliance rule that checks for use of a single control file.

### Example: Sample Rule 2

```
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_rule2">
    <DisplayName nlsid="SAMPLE_RULE_2_NAME">Sample Rule 2</DisplayName>
    <TargetType>oracle_database</TargetType>
    <IsSystem>true</IsSystem>
    <TargetPropertyFilter>
        <PropertyItem>
            <PropertyName>orcl_gtp_operating_system</PropertyName>
            <ValueList>
                <Value>Windows</Value>
            </ValueList>
        </PropertyItem>
        <PropertyItem>
            <PropertyName>orcl_gtp_target_version</PropertyName>
            <ValueList>
                <Value>8.1.6+</Value>
            </ValueList>
        </PropertyItem>
    </TargetPropertyFilter>
    <Description nlsid="SAMPLE_RULE_2_DESC">Checks for use of a single control
file</Description>
    <Impact nlsid="SAMPLE_RULE_2_IMPACT">The control file is one of the most
important files in an Oracle database.
It maintains many physical characteristics and important recovery information
about the database. If you lose the only copy of the control file due to a media
error, there will be unnecessary down time and other risks.</Impact>
    <Recommendation nlsid="SAMPLE_RULE_2_RECO">Use at least two control files
that are multiplexed on different disks.</Recommendation>
    <ViolationContextList>
        <Column type="String" name="FILE_LIST">
            <DisplayLabel nlsid="SAMPLE_RULE_2_COL_1">FILE_LIST</DisplayLabel>
            <IsHidden>false</IsHidden>
            <IsKey>false</IsKey>
        </Column>
        <Column type="Number" name="CONTROL_FILE_COUNT">
            <DisplayLabel nlsid="SAMPLE_RULE_2_COL_2">CONTROL_FILE_COUNT</
```

```
DisplayLabel>
            <IsHidden>false</IsHidden>
            <IsKey>false</IsKey>
        </Column>
    </ViolationContextList>
    <CheckSource>
        <RepositoryCheckDefinition>
            <Metric>
    <TargetType>oracle_database</TargetType>
    <MetricName>sample_rule2</MetricName>
    <SourceType>SQL</SourceType>
        <Source>select CONTROL_FILE_COUNT, FILE_LIST, TARGET_GUID from
MGMT$CS_DB_CONTROL_FILE_COUNT</Source>
    <MetricColumnList>
        <MetricColumnInfo>
            <ColumnName>FILE_LIST</ColumnName>
            <ColumnType>String</ColumnType>
            <isKey>false</isKey>
            <ColumnLabel nlsid="SAMPLE_RULE_2_COL_1">FILE_LIST</ColumnLabel>
                </MetricColumnInfo>
        <MetricColumnInfo>
            <ColumnName>CONTROL_FILE_COUNT</ColumnName>
            <ColumnType>Number</ColumnType>
            <isKey>false</isKey>
            <ColumnLabel nlsid="SAMPLE_RULE_2_COL_2">CONTROL_FILE_COUNT</
ColumnLabel>
        </MetricColumnInfo>
    </MetricColumnList>
            </Metric>
        <ParameterList>
            <RuleParameter>
                <ParamName>CONTROL_FILE_COUNT</ParamName>
                <ParamType>Number</ParamType>
            </RuleParameter>
        </ParameterList>
        <ParameterDefaultSettings>
            <ParamValue>
                <ParamName>CONTROL_FILE_COUNT</ParamName>
                <MinorWarnThreshold>1</MinorWarnThreshold>
            </ParamValue>
        </ParameterDefaultSettings>
        <TestCondition>
            <ThresholdCriteria>
                <ColumnName>CONTROL_FILE_COUNT</ColumnName>
                <TestOperator>EQ</TestOperator>
                <ThresholdValue>1</ThresholdValue>
                <ThresholdType>Number</ThresholdType>
            </ThresholdCriteria>
        </TestCondition>
    </RepositoryCheckDefinition>
    </CheckSource>
  <Severity>MinorWarning</Severity>
  <LifeCycleStatus>Production</LifeCycleStatus>
  <KeywordList>
      <Keyword nlsid="CONFIGURATION">Configuration</Keyword>
  </KeywordList>
  <ViolationMessage nlsid="SAMPLE_RULE_2_VIOL_MSG">The database has an
insufficient number of control files.</ViolationMessage>
  <ClearViolationMessage nlsid="SAMPLE_RULE_2_VIOL_CLEAR_MSG">The database has
sufficient number of control files.</ClearViolationMessage>
```

```
    <Author>SYSMAN</Author>
</Rule>
```

The following example provides an example of a compliance rule that checks for use of a single control file.

**Example: Sample Rule 3**

```
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_rule3">
    <DisplayName nlsid="SAMPLE_RULE_3_NAME">Sample Rule 3</DisplayName>
    <TargetType>oracle_database</TargetType>
    <IsSystem>true</IsSystem>
    <TargetPropertyFilter>
        <PropertyItem>
            <PropertyName>orcl_gtp_operating_system</PropertyName>
            <ValueList>
                <Value>Windows</Value>
            </ValueList>
        </PropertyItem>
        <PropertyItem>
            <PropertyName>orcl_gtp_target_version</PropertyName>
            <ValueList>
                <Value>8.1.6+</Value>
            </ValueList>
        </PropertyItem>
    </TargetPropertyFilter>
    <Description nlsid="SAMPLE_RULE_3_DESC">Checks for use of a single control
file</Description>
    <Impact nlsid="SAMPLE_RULE_3_IMPACT">The control file is one of the most
important files in an Oracle database.
It maintains many physical characteristics and important recovery information
about the database. If you lose the only copy of the control file due to a media
error, there will be unnecessary down time and other risks.</Impact>
    <Recommendation nlsid="SAMPLE_RULE_3_RECO">Use at least two control files
that are multiplexed on different disks.</Recommendation>
    <ViolationContextList>
        <Column type="String" name="FILE_LIST">
            <DisplayLabel nlsid="SAMPLE_RULE_3_COL_1">FILE_LIST</DisplayLabel>
            <IsHidden>false</IsHidden>
            <IsKey>false</IsKey>
        </Column>
        <Column type="Number" name="CONTROL_FILE_COUNT">
            <DisplayLabel nlsid="SAMPLE_RULE_3_COL_2">CONTROL_FILE_COUNT</
DisplayLabel>
            <IsHidden>false</IsHidden>
            <IsKey>false</IsKey>
        </Column>
    </ViolationContextList>
    <CheckSource>
        <RepositoryCheckDefinition>
            <Metric>
    <TargetType>oracle_database</TargetType>
    <MetricName>sample_rule3</MetricName>
    <SourceType>SQL</SourceType>
        <Source>select CONTROL_FILE_COUNT, FILE_LIST, TARGET_GUID from
MGMT$CS_DB_CONTROL_FILE_COUNT</Source>
    <MetricColumnList>
        <MetricColumnInfo>
                <ColumnName>FILE_LIST</ColumnName>
                <ColumnType>String</ColumnType>
```

```
                    <isKey>false</isKey>
                    <ColumnLabel nlsid="SAMPLE_RULE_3_COL_1">FILE_LIST</ColumnLabel>
            </MetricColumnInfo>
            <MetricColumnInfo>
                    <ColumnName>CONTROL_FILE_COUNT</ColumnName>
                    <ColumnType>Number</ColumnType>
                    <isKey>false</isKey>
                    <ColumnLabel nlsid="SAMPLE_RULE_3_COL_2">CONTROL_FILE_COUNT</
ColumnLabel>
            </MetricColumnInfo>
        </MetricColumnList>
                </Metric>
            <ParameterList>
                <RuleParameter>
                    <ParamName>CONTROL_FILE_COUNT</ParamName>
                    <ParamType>Number</ParamType>
                </RuleParameter>
            </ParameterList>
            <ParameterDefaultSettings>
                <ParamValue>
                    <ParamName>CONTROL_FILE_COUNT</ParamName>
                    <MinorWarnThreshold>1</MinorWarnThreshold>
                </ParamValue>
            </ParameterDefaultSettings>
            <TestCondition>
                <ThresholdCriteria>
                    <ColumnName>CONTROL_FILE_COUNT</ColumnName>
                    <TestOperator>EQ</TestOperator>
                    <ThresholdValue>1</ThresholdValue>
                    <ThresholdType>Number</ThresholdType>
                </ThresholdCriteria>
            </TestCondition>
        </RepositoryCheckDefinition>
    </CheckSource>
  <Severity>MinorWarning</Severity>
  <LifeCycleStatus>Production</LifeCycleStatus>
  <KeywordList>
      <Keyword nlsid="CONFIGURATION">Configuration</Keyword>
  </KeywordList>
  <ViolationMessage nlsid="SAMPLE_RULE_3_VIOL_MSG">The database has an
insufficient number of control files.</ViolationMessage>
  <ClearViolationMessage nlsid="SAMPLE_RULE_3_VIOL_CLEAR_MSG">The database has
sufficient number of control files.</ClearViolationMessage>
  <Author>SYSMAN</Author>
</Rule>
```

The following example provides an example of a compliance rule that checks that no unintended ports are left open.

### Example: Sample Rule 4

```
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_rule4">
    <DisplayName nlsid="SAMPLE_RULE_4_NAME">Sample Rule 4</DisplayName>
    <TargetType>host</TargetType>
    <IsSystem>true</IsSystem>
    <Description nlsid="SAMPLE_RULE_4_DESC">Ensure that no unintended ports are
left open</Description>
    <Impact nlsid="SAMPLE_RULE_4_IMPACT">Open ports may allow a malicious user
to take over the host.</Impact>
    <Recommendation nlsid="SAMPLE_RULE_4_RECOMM">Do not open insecure ports.</
```

```
Recommendation>
    <ViolationContextList>
            <Column type="Number" name="port">
                    <DisplayLabel nlsid="SAMPLE_RULE_4_PORT_COL">Port Number</
DisplayLabel>
                    <IsHidden>false</IsHidden>
                    <IsKey>true</IsKey>
            </Column>
    </ViolationContextList>
    <CheckSource>
        <RepositoryCheckDefinition>
            <Metric>
                    <TargetType>host</TargetType>
                <MetricName>sample_rule4</
MetricName>
                    <SourceType>SQL</SourceType>
                    <Source>SELECT target_guid, port as port, port as dummy FROM
MGMT$ESM_PORTS_LATEST</Source>
                    <MetricColumnList>
                        <MetricColumnInfo>
                            <ColumnName>port</ColumnName>
                            <ColumnType>Number</ColumnType>
                            <isKey>true</isKey>
                            <ColumnLabel nlsid="SAMPLE_RULE_4_LABEL">Port
Number</ColumnLabel>
                        </MetricColumnInfo>
                    </MetricColumnList>
            </Metric>
            <ParameterList>
                <RuleParameter>
                    <ParamName
nlsid="SAMPLE_RULE_4_DFLT_PORT_PNAME">DFLT_PORT</ParamName>
                    <ParamType>Number</ParamType>
                </RuleParameter>
            </ParameterList>
            <ParameterDefaultSettings>
                <ParamValue>
                    <ParamName>DFLT_PORT</ParamName>
                    <MinorWarnThreshold>655</MinorWarnThreshold>
                </ParamValue>
            </ParameterDefaultSettings>
            <TestCondition>
                <SqlWhereClauseCriteria>
                    <WhereClause>:port &lt; :DFLT_PORT</WhereClause>
                </SqlWhereClauseCriteria>
            </TestCondition>
        </RepositoryCheckDefinition>
    </CheckSource>
    <Severity>Critical</Severity>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <KeywordList>
        <Keyword nlsid="SECURITY">Security</Keyword>
    </KeywordList>
    <ViolationMessage nlsid="SAMPLE_RULE_4_MESG">The host is in an insecure
state. Port %port% is open.</ViolationMessage>
    <ClearViolationMessage nlsid="SAMPLE_RULE_4_CLR_MESG">Port %port% is not
open.</ClearViolationMessage>
    <Author>ORACLE</Author>
    <LastUpdatedBy>&lt;SYSTEM&gt;</LastUpdatedBy>
</Rule>
```

The following example provides an example of a compliance rule that checks that no unintended ports are left open.

**Example: Sample Rule 5**

```xml
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_rule5">
    <DisplayName nlsid="SAMPLE_RULE_5_NAME">Sample Rule 5</DisplayName>
    <TargetType>host</TargetType>
    <IsSystem>true</IsSystem>
    <Description nlsid="SAMPLE_RULE_5_DESC">Ensure that no unintended ports are
left open</Description>
    <Impact nlsid="SAMPLE_RULE_5_IMPACT">Open ports may allow a malicious user
to take over the host.</Impact>
    <Recommendation nlsid="SAMPLE_RULE_5_RECOMM">Do not open insecure ports.</
Recommendation>
    <ViolationContextList>
            <Column type="Number" name="port">
                <DisplayLabel nlsid="SAMPLE_RULE_5_PORT_COL">Port Number</
DisplayLabel>
                <IsHidden>false</IsHidden>
                <IsKey>true</IsKey>
            </Column>
    </ViolationContextList>
    <CheckSource>
        <RepositoryCheckDefinition>
            <Metric>
                <TargetType>host</TargetType>
            <MetricName>sample_rule5</
MetricName>
                <SourceType>SQL</SourceType>
                <Source>SELECT target_guid, port as port, port as dummy FROM
MGMT$ESM_PORTS_LATEST</Source>
                <MetricColumnList>
                    <MetricColumnInfo>
                        <ColumnName>port</ColumnName>
                        <ColumnType>Number</ColumnType>
                        <isKey>true</isKey>
                        <ColumnLabel nlsid="SAMPLE_RULE_5_LABEL">Port Number</
ColumnLabel>
                    </MetricColumnInfo>
                </MetricColumnList>
            </Metric>
            <ParameterList>
                <RuleParameter>
                    <ParamName nlsid="SAMPLE_RULE_5_DFLT_PORT_PNAME">DFLT_PORT</
ParamName>
                    <ParamType>Number</ParamType>
                </RuleParameter>
            </ParameterList>
            <ParameterDefaultSettings>
                <ParamValue>
                    <ParamName>DFLT_PORT</ParamName>
                    <MinorWarnThreshold>655</MinorWarnThreshold>
                </ParamValue>
            </ParameterDefaultSettings>
            <TestCondition>
                <SqlWhereClauseCriteria>
                    <WhereClause>:port &lt; :DFLT_PORT</WhereClause>
                </SqlWhereClauseCriteria>
            </TestCondition>
```

```
            </RepositoryCheckDefinition>
    </CheckSource>
    <Severity>Critical</Severity>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <KeywordList>
        <Keyword nlsid="SECURITY">Security</Keyword>
    </KeywordList>
    <ViolationMessage nlsid="SAMPLE_RULE_5_MESG">The host is in an insecure
state. Port %port% is open.</ViolationMessage>
    <ClearViolationMessage nlsid="SAMPLE_RULE_5_CLR_MESG">Port %port% is not
open.</ClearViolationMessage>
    <Author>ORACLE</Author>
    <LastUpdatedBy>&lt;SYSTEM&gt;</LastUpdatedBy>
</Rule>
```

The following example provides an example of a compliance rule that checks that no unintended ports are left open.

**Example: Sample Rule 6**

```
<Rule xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_rule6">
    <DisplayName nlsid="SAMPLE_RULE_6_NAME">Sample Rule 6</DisplayName>
    <TargetType>host</TargetType>
    <IsSystem>true</IsSystem>
    <Description nlsid="SAMPLE_RULE_6_DESC">Ensure that no unintended ports are
left open</Description>
    <Impact nlsid="SAMPLE_RULE_6_IMPACT">Open ports may allow a malicious user
to take over the host.</Impact>
    <Recommendation nlsid="SAMPLE_RULE_6_RECOMM">Do not open insecure ports.</
Recommendation>
    <ViolationContextList>
            <Column type="Number" name="port">
                <DisplayLabel nlsid="SAMPLE_RULE_6_PORT_COL">Port Number</
DisplayLabel>
                <IsHidden>false</IsHidden>
                <IsKey>true</IsKey>
            </Column>
    </ViolationContextList>
    <CheckSource>
        <RepositoryCheckDefinition>
            <Metric>
                <TargetType>host</TargetType>
            <MetricName>sample_rule6</
MetricName>
                <SourceType>SQL</SourceType>
                <Source>SELECT target_guid, port as port, port as dummy FROM
MGMT$ESM_PORTS_LATEST</Source>
                <MetricColumnList>
                    <MetricColumnInfo>
                        <ColumnName>port</ColumnName>
                        <ColumnType>Number</ColumnType>
                        <isKey>true</isKey>
                        <ColumnLabel nlsid="SAMPLE_RULE_6_LABEL">Port Number</
ColumnLabel>
                    </MetricColumnInfo>
                </MetricColumnList>
            </Metric>
            <ParameterList>
                <RuleParameter>
                    <ParamName nlsid="SAMPLE_RULE_6_DFLT_PORT_PNAME">DFLT_PORT</
```

```
ParamName>
                    <ParamType>Number</ParamType>
                </RuleParameter>
            </ParameterList>
            <ParameterDefaultSettings>
                <ParamValue>
                    <ParamName>DFLT_PORT</ParamName>
                    <MinorWarnThreshold>655</MinorWarnThreshold>
                </ParamValue>
            </ParameterDefaultSettings>
            <TestCondition>
                <SqlWhereClauseCriteria>
                    <WhereClause>:port &lt; :DFLT_PORT</WhereClause>
                </SqlWhereClauseCriteria>
            </TestCondition>
        </RepositoryCheckDefinition>
    </CheckSource>
    <Severity>Critical</Severity>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <KeywordList>
        <Keyword nlsid="SECURITY">Security</Keyword>
    </KeywordList>
    <ViolationMessage nlsid="SAMPLE_RULE_6_MESG">The host is in an insecure
state. Port %port% is open.</ViolationMessage>
    <ClearViolationMessage nlsid="SAMPLE_RULE_6_CLR_MESG">Port %port% is not
open.</ClearViolationMessage>
    <Author>ORACLE</Author>
    <LastUpdatedBy>&lt;SYSTEM&gt;</LastUpdatedBy>
</Rule>
```

The following example provides an example of a compliance standard that includes rules to check for use of a single control file.

**Example: Sample Compliance Standard 1**

```
<Standard xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_cs1">
    <DisplayName nlsid="SAMPLE_CS_1_NAME">Sample Compliance Standard 1</
DisplayName>
    <TargetType>oracle_database</TargetType>
        <TargetPropertyFilter>
            <PropertyItem>
                <PropertyName>orcl_gtp_target_version</PropertyName>
                <ValueList>
                    <Value>Windows</Value>
                </ValueList>
            </PropertyItem>
            <PropertyItem>
                <PropertyName>orcl_gtp_target_version</PropertyName>
                <ValueList>
                    <Value>8.1.6+</Value>
                </ValueList>
            </PropertyItem>
        </TargetPropertyFilter>
    <Author>SYSTEM</Author>
    <Version>1</Version>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <IsSystem>true</IsSystem>
    <Description nlsid="SAMPLE_CS_1_DESC">Sample Description</Description>
    <KeywordList>
        <Keyword nlsid="CONFIGURATION">Configuration</Keyword>
```

```
        </KeywordList>
    <ReferenceURL>http://sampleurl.com</ReferenceURL>
    <Body>
            <RuleFolder name="sample_RF_1">
                <DisplayName nlsid="SAMPLE_RF_1_NAME">Sample Rulefolder</
DisplayName>
                <Description nlsid="SAMPLE_RF_1_DESC">This includes rules that
checks for use of a single control file</Description>
                <ReferenceURL>http://www.oracle.com/db_rf1</ReferenceURL>
                <Importance>Normal</Importance>
                <RuleReference>
                <Name>sample_rule1</Name>
                <TargetType>oracle_database</TargetType>
                <Importance>Normal</Importance>
                </RuleReference>
            </RuleFolder>
    </Body>
</Standard>
```

The following example provides an example of a compliance standard that includes rules to check for open unsecured ports.

### Example: Sample Compliance Standard 2

```
<Standard xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_cs2">
    <DisplayName nlsid="SAMPLE_CS_2_NAME">Sample Compliance Standard 2</
DisplayName>
    <TargetType>host</TargetType>
    <Author>SYSTEM</Author>
    <Version>1</Version>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <IsSystem>true</IsSystem>
    <Description nlsid="SAMPLE_CS_2_DESC">Sample Description</Description>
    <KeywordList>
        <Keyword nlsid="SECURITY">Security</Keyword>
    </KeywordList>
    <ReferenceURL>http://sampleurl.com</ReferenceURL>
    <Body>
        <RuleFolder name="sample_RF_2">
            <DisplayName nlsid="SAMPLE_RF_2_NAME">Sample Rulefolder</DisplayName>
            <Description nlsid="SAMPLE_RF_2_DESC">This includes rules that
checks for open insecure ports.</Description>
            <ReferenceURL>http://www.oracle.com/db_rf1</ReferenceURL>
                            <Importance>Normal</Importance>
                            <RuleReference>
                            <Name>sample_rule4</Name>
            <TargetType>host</TargetType>
            <Importance>Normal</Importance>
                            </RuleReference>
        </RuleFolder>
    </Body>
</Standard>
```

The following example provides an example of a compliance standard that includes rules to check for open unsecured ports.

### Example: Sample Compliance Standard 3

```
<Standard xmlns="http://www.oracle.com/DataCenter/ConfigStd"
oms_version="12.1.0.1.0" name="sample_cs3">
    <DisplayName nlsid="SAMPLE_CS_3_NAME">Sample Compliance Standard 3</
```

```
DisplayName>
    <TargetType>host</TargetType>
    <Author>SYSTEM</Author>
    <Version>1</Version>
    <LifeCycleStatus>Production</LifeCycleStatus>
    <IsSystem>true</IsSystem>
    <Description nlsid="SAMPLE_CS_3_DESC">Sample Description</Description>
    <KeywordList>
        <Keyword nlsid="SECURITY">Security</Keyword>
    </KeywordList>
    <ReferenceURL>http://sampleurl.com</ReferenceURL>
    <Body>
            <RuleFolder name="sample_RF_3">
                <DisplayName nlsid="SAMPLE_RF_3_NAME">Sample Rulefolder</
DisplayName>
                <Description nlsid="SAMPLE_RF_3_DESC">This includes rules that
checks for open insecure ports.</Description>
                <ReferenceURL>http://www.oracle.com/db_rf1</ReferenceURL>
                <Importance>Normal</Importance>
                <RuleReference>
                <Name>sample_rule5</Name>
                <TargetType>host</TargetType>
                <Importance>Normal</Importance>
                </RuleReference>
            </RuleFolder>
    </Body>
</Standard>
]]
```

The following example provides an example of a compliance framework.

**Example: Sample Compliance Framework**

```
<StandardGroup xmlns="http://www.oracle.com/DataCenter/ConfigStd"
name="sample_csg" oms_version="12.1.0.1.0">
        <DisplayName nlsid="SAMPLE_CSG_NAME">Sample Compliance Framework</
DisplayName>
        <Author>SYSTEM</Author>
        <Version>1</Version>
        <LifeCycleStatus>Production</LifeCycleStatus>
        <Description nlsid="SAMPLE_CSG_DESC">Sample Description</Description>
        <KeywordList>
            <Keyword nlsid="SECURITY">Security</Keyword>
        </KeywordList>
        <ReferenceURL>http://sampleurl.com</ReferenceURL>
        <IsHidden>false</IsHidden>
        <IsSystem>true</IsSystem>
        <GroupBody>
            <SubGroup name="SampleSubgroup">
                <DisplayName nlsid="SAMPLE_CSG_SUBGROUP_NAME">Sample Child
Framework</DisplayName>
                <Description nlsid="SAMPLE_CSG_SUBGROUP_DESC">Sample Child
framework Description</Description>
                <ReferenceURL>http://sampleurl.com</ReferenceURL>
                <Importance>Normal</Importance>
                <StandardReference>
                    <Name>sample_cs3</Name>
                    <Author>SYSTEM</Author>
                    <Version>1</Version>
                    <Importance>Normal</Importance>
                </StandardReference>
```

```
            </SubGroup>
          </GroupBody>
   </StandardGroup>
```

The following example provides an example of compliance content.

**Example: Compliance Content Version 2**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE ComplianceContent [
<!ENTITY rule1 SYSTEM "SampleRule1.xml">
<!ENTITY rule2 SYSTEM "SampleRule2.xml">
<!ENTITY rule3 SYSTEM "SampleRule3.xml">
<!ENTITY rule5 SYSTEM "SampleRule5.xml">
<!ENTITY rule6 SYSTEM "SampleRule6.xml">
<!ENTITY standard1 SYSTEM "SampleComplianceStandard1.xml">
<!ENTITY standard3 SYSTEM "SampleComplianceStandard3.xml">
<!ENTITY standardgroup SYSTEM "SampleComplianceFramework.xml">
]>
<ComplianceContent xmlns="http://www.oracle.com/DataCenter/ConfigStd"
name="SampleComplianceContent" oms_version="12.1.0.1.0"
content_version="12.1.0.2.0">
<ChangeList>
  <!-- ChangeList tag process each of the Change Tag with respect to the version
of the ComplianceContent installed in repository. -->

    <Change version="12.1.0.1.0">

    <!-- AddSubGroupWithinStandardGroup/AddStandardReferenceToStandardGroup tags
will modify StandardGroup definition. -->
    <!-- AddSubGroupWithinStandardGroup will introduce a subgroup within an
existing  compliance framework/standard group in repository. -->
    <!-- AddStandardReferenceToStandardGroup will introduce a reference to a
standard within an existing  compliance framework/standard group in repository.
-->

        <AddSubGroupWithinStandardGroup order="2">
            <StandardGroupName>oracle_pci</StandardGroupName>
            <StandardGroupAuthor>ORACLE</StandardGroupAuthor>
            <StandardGroupVersion>1</StandardGroupVersion>
            <SubGroup name="sampleSubgroup1">
                <DisplayName>sub1</DisplayName>
                <ReferenceURL>http://sampleAddedSubgroup.com</ReferenceURL>
                <Importance>High</Importance>
            </SubGroup>
        </AddSubGroupWithinStandardGroup>
        <AddStandardReferenceToStandardGroup>
            <StandardGroupName>oracle_pci</StandardGroupName>
            <StandardGroupAuthor>ORACLE</StandardGroupAuthor>
            <StandardGroupVersion>1</StandardGroupVersion>
            <SubGroupListInfo>
                <SubGroupElem>oracle_pci_ctrlobj_a</SubGroupElem>
            </SubGroupListInfo>
            <StandardReference>
                <Name>sample_cs1</Name>
                <Author>SYSTEM</Author>
                <Version>1</Version>
            </StandardReference>
        </AddStandardReferenceToStandardGroup>
        </Change>
```

```
        <Change version="12.1.0.2.0">

            <!-- Delete will be remove rule/standard from repository if present,
else it will be noop. -->

        <DeleteStandard>
            <StandardName>sample_cs2</StandardName>
            <StandardAuthor>SYSTEM</StandardAuthor>
            <StandardVersion>1</StandardVersion>
        </DeleteStandard>

        <DeleteRule>
            <RuleName>sample_rule4</RuleName>
            <TargetType>host</TargetType>
        </DeleteRule>


        <!-- Entities with Update tag will override definitions if they exist in
the repository. -->
        <!-- Please note that if standard/rule is updated then old results are
            replaced by new results based on standard/rule definition after
update.  -->

        <UpdateRule>
            <RuleName>sample_rule5</RuleName>
            <TargetType>host</TargetType>
        </UpdateRule>
        <UpdateStandard>
            <StandardName>sample_cs3</StandardName>
            <StandardAuthor>SYSTEM</StandardAuthor>
            <StandardVersion>1</StandardVersion>
        </UpdateStandard>

        <UpdateStandardGroup>
            <StandardGroupName>sample_csg</StandardGroupName>
            <StandardGroupAuthor>SYSTEM</StandardGroupAuthor>
            <StandardGroupVersion>1</StandardGroupVersion>
        </UpdateStandardGroup>

        <!-- AddSubGroupWithinStandardGroup will introduce a subgroup within an
existing  compliance framework/standard group in repository. -->
        <!-- AddStandardReferenceToStandardGroup will introduce a reference to a
standard within an existing  compliance framework/standard group in repository.
-->

        <AddSubGroupWithinStandardGroup order="2">
            <StandardGroupName>oracle_pci</StandardGroupName>
            <StandardGroupAuthor>ORACLE</StandardGroupAuthor>
            <StandardGroupVersion>1</StandardGroupVersion>
            <SubGroup name="sampleSubgroup2">
                    <DisplayName>sub2</DisplayName>
                    <ReferenceURL>http://sampleAddedSubgroup.com</ReferenceURL>
                    <Importance>High</Importance>
            </SubGroup>
        </AddSubGroupWithinStandardGroup>
        <AddStandardReferenceToStandardGroup>
            <StandardGroupName>oracle_pci</StandardGroupName>
            <StandardGroupAuthor>ORACLE</StandardGroupAuthor>
            <StandardGroupVersion>1</StandardGroupVersion>
            <SubGroupListInfo>
```

```
                    <SubGroupElem>oracle_pci_ctrlobj_a</SubGroupElem>
            </SubGroupListInfo>
            <StandardReference>
                    <Name>sample_cs3</Name>
                    <Author>SYSTEM</Author>
                    <Version>1</Version>
            </StandardReference>
        </AddStandardReferenceToStandardGroup>
        </Change>

</ChangeList>
<!--List of compliance standard rules -->
&rule1;
&rule2;
&rule3;
&rule5;
&rule6;
<!--List of compliance standards -->
&standard1;
&standard3;
<!--List of compliance standard groups/frameworks -->
&standardgroup;
</ComplianceContent>
```

# Publishing Compliance Content Using Self Update

If you want to publish compliance content without having to deploy the plug-in, than use the Self Update console.

To publish and apply compliance content from the Self Update console:

1. Create a compliance content JAR file from the XML content using the following command:

   ```
   -jar cvfM compliancecontent.jar compliance_content_files
   ```

   > **Note:**
   >
   > Similarly, multiple DLF files can be combined in a JAR file.

2. Create a manifest file to specify the name of the compliance content, label, and the version of the compliance content to be published. This manifest file specifies compliancecontent.jar and compliancedlf.jar in order respectively.

   **Example: Sample Manifest File**

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <tns:EntityInstance xmlns:tns="http://www.oracle.com/EnterpriseGridControl/
   SelfUpdateManifest"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" EntityType="param"
   EntityTypeVersion="12.1.0.1.0" Vendor="Oracle" Maturity="TEST">

   <tns:Description><![CDATA[<param>]]></tns:Description>

   <tns:AttributeList>
   <tns:Version>12.1.0.1.0</tns:Version>
   <tns:Attribute Name="name" Value="<param>" Label="display_name"/>
   </tns:AttributeList>
   ```

```
<tns:Readme><![CDATA[ <param>]]>
</tns:Readme>
<tns:CustomParamList/>
<tns:DependsOn/>
<tns:ArchiveList>
<tns:Archive Filename="param"/>
<tns:Archive Filename="param"/>
</tns:ArchiveList>
<tns:CustomData/>
</tns:EntityInstance>
```

3. Create a SAR (self update archive) file from the manifest file, compliancecontent.jar, and compliancedlf.jar using the following command:

```
edkutil prepare_update -manifest manifest_file_name -archivedir
directory_containing_compliancecontent.jar_and_compliancedlf.jar -out
sar_file_name
```

> **Note:**
>
> Before you import the SAR file into Enterprise Manager, make sure that the Software Library is configured.
>
> For more information, see Setting Up the Software Library.

4. Import the SAR file into Enterprise Manager using the following command:

```
emcli import_update -omslocal -file=complete_path_to_sar_file
```

> **Note:**
>
> Using the `-omslocal` flag means that the file must be placed on the Oracle Management Server (OMS) file system.

5. Log in to Enterprise Manager. From the **Setup** menu, select **Extensibility**, and then **Self Update**.

   The Self Update page appears.

6. From the **Status** area, check that Downloaded Updates is set to 1 for Compliance Content.

7. In the Type column, click **Compliance Content**.

   The Self Update: Compliance Content page appears.

8. Select the row with downloaded in the Status column, then **Apply**. Follow the steps in the wizard that appears.

9. From the Actions list, select **Apply** and check that the Status column reads succeeded.

10. Verify the imported compliance content from the Compliance Library. To view the Compliance Library, from the **Enterprise** menu, select **Compliance**, then select **Library**.

# 14

# Validating, Packaging, and Deploying the Plug-in

This chapter contains the following sections:

## Introduction to Validating, Packaging, and Deploying the Plug-in

As a plug-in developer, you are responsible for the following steps within the validation, packaging, and deployment process:

1. Create the staging directory (*plugin_stage*):

   The staging directory structure defines the location of files as they will be deployed to Oracle Management Service and Management Agents.

   For more information, see Staging the Plug-in.

2. Validate the plug-in.

   Use the `empdk validate_plugin` command to validate the content of the plug-in once you have designed and developed it. This command verifies that the XML metadata files are compliant.

   For more information, see Validating the Plug-in.

3. Create the Oracle Plug-in Archive (OPAR) file.

   The plug-in archive is the standard way of distributing the plug-in for importing and deploying the plug-in across different installations of the Enterprise Manager Cloud Control.

   For more information, see Creating the Plug-in Archive.

4. Import the OPAR into Enterprise Manager.

   Use the `emcli import_update` command to import the plug-in into Enterprise Manager.

   For more information, see Importing and Deploying the Plug-in Archive into Enterprise Manager.

5. Deploy the plug-in.

   You must deploy a plug-in on the Oracle Management Service before it is used for monitoring targets.

   For more information, see Deploying the Plug-in on Oracle Management Service (OMS).

6. Add target instances for the deployed plug-in to monitor.

   For more information, see Adding a Target Instance.

7. Use the Metadata Registration Service (MRS) to deploy updated metadata files.

   The MRS allows you to upload one or more updated metadata files to the Oracle Management Service and Management Agents where your plug-in is deployed. The MRS registers the updated metadata files with Enterprise Manager, and overwrites the existing metadata with your updates.

   For more information, see Updating Deployed Metadata Files Using the Metadata Registration Service (MRS).

# Staging the Plug-in

After you have created the plug-in files, the next step is to stage the plug-in in preparation for validation and packaging. The staging directory structure defines the location of files as they will be deployed to Oracle Management Service and Management Agents.

The following example provides an example of the staging directory structure and Table 14-1 describes the archive directory structure. Files are placed in the archive based on whether they are deployed to Oracle Management Service, Management Agents, or both. When the plug-in is deployed to an OMS instance or a Management Agent, the requisite files are copied to their respective directory locations.

**Example: Plug-in Directory Structure**

```
plugin_stage/
          |
        plugin.xml
        agent/
              |
              plugin_registry.xml
              default_collection/
                               |
                               target_type.xml
              metadata/
                     |
                     target_type.xml
              scripts/
                     |
                     scripts
        oms/
           |
          metadata/
                 |
                 default_collection/
                                  |
```

```
                                          target_type.xml
                    derivedAssoc/
                            |
                               derivedAssoc_rule.xml
                    discovery/
                         |
                           discovery.xml
                    gccompliance/
                              |
                                 ComplianceContent_name.xml
                    jobTypes/
                          |
                             job_type.xml
                    mpcui/
                        |
                         mpcui.xml
                    reports/
                          |
                            report.xml
                    snapshotlive/
                             |
                               target-type_ecmdef.xml
                    targetType/
                            |
                              target_type.xml
          discovery/
                 |
                    discovery scripts
```

> **Note:**
>
> Use of the specified subdirectory names within the archive are not required, but are recommended by Oracle.

**Table 14-1    File Locations in Plug-in Archive Structure**

| File | Directory | Notes |
|------|-----------|-------|
| plugin.xml | plugin_stage/ | Required. This file defines generic plug-in metadata that is deployed to Oracle Management Service. Place it at the root level within the archive structure.For more information, see Creating the plugin.xml File. |
| plugin_registry.xml | plugin_stage/agent/ | Required. This file defines metadata describing the plug-in used by the Management Agent. It must be placed at the top level of the /agent directory. For more information, see Creating the plugin_registry.xml File. |

**Table 14-1    (Cont.) File Locations in Plug-in Archive Structure**

| File | Directory | Notes |
| --- | --- | --- |
| *target_type*.xml | *plugin_stage*/oms/metadata/targetType/<br>*plugin_stage*/agent/metadata/ | Required. This file defines metrics to be collected or computed for the target type.<br><br>An identical copy of this file must be placed in both the /oms and /agent directories. For more information, see Creating the Target Type Metadata File. |
| *default_collections*.xml | *plugin_stage*/oms/metadata/default_collection/<br>*plugin_stage*/agent/default_collection/ | Required. This file defines metric collection parameters such as metric data collection frequency and default metric alert thresholds.<br><br>An identical copy of this file must be placed in both the /oms and /agent directories. For more information, see Creating the Default Collection File.<br><br>**Note**: Oracle recommends that you name the default collections metadata file with the same file name as the target type metadata file. |
| *target-type*_ecmdef.xml | *plugin_stage*/oms/metadata/snapshotlive/ | Optional. Defines configuration data collection. For more information, see Defining Configuration Collection Tables. |
| *job_type*.xml | *plugin_stage*/oms/metadata/jobTypes/ | Optional. Place all job type definition files in the /jobTypes directory. For more information, see Adding Job Types . |
| *report*.xml | *plugin_stage*/oms/metadata/reports/ | Optional. Put report definition files in the /reports directory. |
| *derivedAssoc_rule*.xml | *plugin_stage*/oms/metadata/derivedAssoc/ | Optional. Place the metadata file that defines the association derivation rules (or set of rules) in this directory. For more information, see Using Derived Associations. |
| *ComplianceContent_name*.xml | *plugin_stage*/oms/metadata/gccompliance/ | Optional. *ComplianceContent_name.xml* contains references to compliance standards, rules, and frameworks. This directory can contain compliance rule, compliance standard, and compliance framework XML files. For more information, see Packaging Compliance XML. |

**Table 14-1  (Cont.) File Locations in Plug-in Archive Structure**

| File | Directory | Notes |
|------|-----------|-------|
| *compliance*.dlf | *plugin_stage*/oms/rsc/*area*/gccompliance/ | Optional. Place all Data Loading Format (DLF) map entry (DLF) files associated with compliance rules or standards definitions in this directory. |
| *mpcui*.xml | *plugin_stage*/oms/metadata/mpcui/ | Optional. Place all management user interface metadata files in this directory. For more information, see Defining a Management User Interface . |
| *discovery*.xml | *plugin_stage*/oms/metadata/discovery/ | Optional. Place discovery metadata files in this location. For more information, see Packaging Discovery XML and Discovery Content. |
| discovery script file(s) | *plugin_stage*/discovery/ | Optional. Place the Perl scripts and JAR files (if any) that are required to perform automatic discovery in this location. For more information, see Packaging Discovery XML and Discovery Content. |
| script file(s) | *plugin_stage*/agent/scripts/ | Optional. Put any scripts that will be deployed to Management Agents, such as metric collection scripts invoked by fetchlets, in this location.<br><br>Use of the /scripts directory is not required but is recommended, as it allows use of the %scriptsDir% token from metric query descriptors defined in the target-type.xml file and in job type command references. |

## Modifying File Permissions Within the Plug-in Directory

To specify customized file permissions for scripts packaged under the plug-in directory:

1. Create the following text file in the relevant directory (oms, agent, or discovery):

   ```
   plugin_stage/oms/file_permissions.txt
   plugin_stage/agent/file_permissions.txt
   plugin_stage/discovery/file_permissions.txt
   ```

2. Specify the permissions for multiple files, each in one row. The format of the text file is similar to the following:

   *file_name*:*file_permission_number*

   or

   *file_pattern*:*file_permission_number*

   In these formats:

- *file_name* represents the name of the file that will have permissions changed

- *file_pattern* represents a pattern that the script will search for, such as a certain file type or all file names containing the pattern. Then all matched files will have permissions changed.

- *file_permission_number* represents the UNIX permission description of the file. This is the new permissions value of the matched files.

For example, to change the permissions of all occurrences of example1.sh to 655 within the *plugin_stage*/oms directory, add the following entry to the *plugin_stage*/oms/file_permissions.txt file:

```
example1.sh:655
```

To change the permissions of all script files (of file type *.sh) to 655 within the *plugin_stage*/discovery directory, add the following entry to the *plugin_stage*/discovery/file_permissions.txt file:

```
*.sh:655
```

To change the permissions of all files beginning with abc to 655 within the *plugin_stage*/agent directory, add the following entry to the *plugin_stage*/agent/file_permissions.txt file:

```
abc*:655
```

3. Save and close the file_permissions.txt file. These permissions are read and retained during plug-in creation with the OPAR and later used during the plug-in software installation. For more information about plug-in creation, see Creating the Plug-in Archive.

# Validating the Plug-in

Validate the plug-in throughout the development cycle, and before packaging the plug-in. Use the `empdk validate_plugin` command to validate the content of the plug-in after you have designed and developed it to verify that the XML metadata files are compliant. The tool is run against the specified plug-in staging directory and generates a report of any violations found. Specify the format of the generated report using the `-format` option.

The following example provides the command usage.

**Example: empdk validate_plugin Command Usage**

```
empdk validate_plugin -stage_dir staging directory
                [-tmp_dir temporary working location]
                [-out_dir output directory]
                [-format (html|text|xml)]
                [-conn_desc] - not used by external developers
                [-repos_user Enterprise Manager repository owner]
                [-debug [file to output debug information to]]
```

The following example validates the plug-in source files in the specified staging directory, and generates the validation report as a text file in the current working directory:

```
edk\bin>empdk validate_plugin -stage_dir C:\plugin_staging -format text
```

Table 14-2 provides the options that can be used to validate the plug-in.

**Table 14-2    Options for Validating the Plug-in**

| Option | Description |
|---|---|
| -tmp_dir | Specify a temporary location to extract the plug-in files into. If not specified, it defaults to the current directory. |
| -out_dir | The directory the validation report file will generated into. If not specified, the report file will be generated in the current working directory. |
| -debug | Specify a file name where you want to store the debug information. If not specified, the default log file (validateplugin.logtime) will be created in the out directory and will store warning and error message only. If specified, then it will store all the debugging information to that log file |
| -format | The format the validation report will be generated in. If not specified, the report will be generated as a text file. |

# Creating the Plug-in Archive

After you have created the plug-in stage directory and validated the plug-in, the next step is to create an Oracle Plug-in Archive (OPAR) file. The OPAR file plays an important role at various stages of the plug-in lifecycle. It serves the following:

- The plug-in archive is the standard way of distributing the plug-in for importing and deploying the plug-in across different installations of the Enterprise Manager Cloud Control.

- You must test the plug-in being developed on an Enterprise Manager Cloud Control installation.

A plug-in is created by adding the files previously discussed to an OPAR using the Enterprise Manager Extensibility Development Kit (EDK). For more information about the EDK, see the *Oracle Enterprise Manager Cloud Control Extensibility Programmer's Guide*.

To create an OPAR, at the command prompt, enter the `empdk create_plugin` command. For more information about the `create_plugin` verb, see the command line help

The `empdk create_plugin` command syntax is as follows:

```
empdk create_plugin -stage_dir staging dir -conn_desc
repository_connection_string -repos_user username [-repos_password
repos_password]
           -out_dir output_directory [-debug] [-force]
```

For example:

```
edk\bin>empdk create_plugin -stage_dir C:\pluginstagdir -conn_desc
myhost.us.example.com:25055:$ORACLE_SID -repos_user sysman -out_dir /tmp/plugins
```

Table 14-3 provides the options that can be used to create an OPAR:

**Table 14-3    Options for Creating an OPAR**

| Option | Description |
| --- | --- |
| `-tmp_dir` | This option enables the command to create a temporary directory while executing. You can specify the path you want to use for this by providing a value following the option. Specify an existing directory or else you will receive an error. If not specified, then the out directory will be used for temporary location. If no out directory is specified, the current directory is the default. |
| `-out_dir` | The directory in which the plug-in archive (*.opar) file will be created. If not specified, the plug-in archive will be created in the current directory. |
| `-debug` | Specify a file name where you want to store the debug information.If not specified, the default log file (createplugin.logtime) will be created in the out directory and will store only warning and error messages. If specified, then it will store all the debugging information to that log file. This debugging information can be used to identify issues you may encounter while creating a plug-in. You can append the log created when you are filing a support request for a create plug-in related issue. |
| `-force` | If the out directory contains an OPAR with the same name, then you will be prompted to specify whether to overwrite the existing OPAR. If provided, it will automatically overwrite the existing OPAR. This is disabled by default. |
| `-conn_desc` | The connection descriptor that will connect to the Management Repository that the plug-in metadata will be written to when the plug-in is imported into Enterprise Manager. Specify the connection descriptor using the following syntax: `host:port:sid` For example: `myhost.us.example.com:25055:$ORACLE_SID` |
| `-repos_user` | The user to connect to the Management Repository. |

If the command runs successfully, then a *plugin_version*.*plugin_id*.opar archive will be created in the directory where you ran this command.

If the command fails, an appropriate error message will be displayed. The parameters passed to the commands vary from user to user and across systems where the plug-in is being created.

Some common mistakes while trying to create the plug-in archive are:

- If the path to the staging directory is entered incorrectly, then it raises a *File not found* or an *Input not found* exception.

- The *empdk command not found* exception will be shown if you have not changed your current directory to the expanded EDK directory.

- If the disk where you are trying to create the OPAR has inadequate memory, then an Input/Output-related exception might occur.

# Importing and Deploying the Plug-in Archive into Enterprise Manager

Once you have the plug-in archive ready with your *.opar file, you must import the plug-in into Enterprise Manager. Importing ensures that the content that you have created and packaged in the plug-in is available within Enterprise Manager.

> **Note:**
>
> You must first import the plug-in before it can be deployed onto Enterprise Manager.

## Prerequisites for Importing the Plug-in

The following prerequisites are met before importing the plug-in.

## Setting Up the Software Library

1. Create a folder in the system where Enterprise Manger is installed. For example, /net/*hostname*/scratch/aime/swlib1.
2. From the console, select **Enterprise**, then **Provisioning and Patching**, and then **Software Library**.
3. Click **Actions**, then **Administration**.
4. Click **Add**.
5. In the pop up window, enter a name and location. For example, `swlib1` and `/net/hostname/scratch/aime/swlib1`. This should be the folder that you created in step 1.
6. Wait for the processing to finish.

## Setting Up the EM CLI Utility

You will use the Enterprise Manager Command Line Utility, or EM CLI, to import the plug-in archive for deployment to Enterprise Manager.

- A page is provided in the Cloud Control console with instructions on setting up EM CLI. Access the page at the following URL:

  ```
  https://em_host:em_port/em/console/emcli/download
  ```

  For example:

  ```
  https://emserver.test.com:7799/em/console/emcli/download
  ```

- After setting up EM CLI, synchronize the EM CLI client with an Oracle Management Service (OMS):

  ```
  emcli sync
  ```

After synchronization, all verbs and associated command line help available to this OMS become available at the EM CLI client.

# Importing the Plug-in Archive

Once packaged, the plug-in must be imported into Enterprise Manager Cloud Control using the `emcli import_update` command. You have two options depending on where EM CLI is installed:

- If EM CLI is on the same system as the system where you created the plug-in archive (*.opar file), then run the following command.

```
emcli import_update
 -file="<path to *.opar file you created>"
 -omslocal
```

The `-omslocal` flag indicates that the plug-in archive is on the same system where you are running this command and the path exists on this system.

For example:

```
emcli import_update -file=/tmp/sample_plugin.opar -omslocal
```

- If you are running EM CLI on a different system than the system where you created the plug-in archive (*.opar file), then run the following command:

```
emcli import_update
           -file="path to the .opar file"
           -host="host name of plug-in host"
           -credential_name="credential for plug-in host"
           -credential_owner="credential owner on the plug-in host"
```

where:

- `-file`: The absolute path to the *.opar file on the system where you created the archive.

- `-host`: The host name for the host target where the file is available.

- `-credential_name`: The name of the credentials on the remote system you are connecting to.

- `-credential_owner`: The owner of the credentials on the host system you are connecting to.

For example:

```
emcli import_update -file=/tmp/sample_plugin.opar
-host="host1.test.com" -credential_name="myOracleCred" -
credential_owner="password"
```

- As an alternative to the previous step, you can also run the following command:

```
emcli import_update
     -file="path to *.opar file you created"
     -host="hostname"
     -credential_set_name="setname"
```

`-credential_set_name`: The set name of the preferred credential stored in the Management Repository for the host target. It can be one of the following:

- `HostCredsNormal`: The default unprivileged credential set.

    &ndash;   `HostCredsPriv`: The privileged credential set.

# Deploying the Plug-in on Oracle Management Service (OMS)

A plug-in must be deployed on Oracle Management Service (OMS) before it can be used to monitor targets. Follow the steps below to deploy the plug-in on Enterprise Manager Cloud Control.

> **Note:**
>
> Plug-ins must be deployed on Oracle Management Service before being deployed on Management Agents.
>
> Plug-ins for specific target types are deployed automatically on Management Agents that will monitor those target types. For more information, see Adding a Target Instance.

To deploy a plug-in on the Oracle Management Server:

1. From the **Setup** menu, select **Extensibility**, then **Plug-ins**.

   Enterprise Manager displays the list of plug-ins that have been downloaded and can be deployed on the Plug-ins page.

2. On the Plug-ins page, select the specific plug-in you want to deploy.

3. Click **Deploy On**, then select **Management Servers**.

   Ensure that dependent plug-ins are deployed and that all existing Management Agents are compatible with the version of the specified plug-in. Enterprise Manager prompts for credentials if the Management Agent is not available.

4. On the **Deploy Plug-in** window, enter the required details. Note that you will require the Management Repository SYS user password to complete the deployment process.

   From the Version list, select the Plug-in version. The **Target Type** information is displayed in the table. Enter the **Repository sys Password**, then click **Continue**.

5. Proceed through the steps in the Deploy Plug-in windows.

6. Click **Deploy** to deploy the selected plug-in on all Enterprise Manager servers.

   Enterprise Manager displays a page that allows you to monitor the deployment status. Enterprise Manager deploys the selected plug-in on all Enterprise Manager Servers.

   You can also monitor the deployment status by going to the Enterprise Manager Cloud Control console, then going to the Plug-ins page as described in step 1, selecting the plug-in and select the **Recent Deployment Activities** tab at the bottom of the page for the selected plug-ins. This bottom section also lets you see details of your plug-in, which includes the plug-in ID, version, vendor, and so on.

   If any of the steps during plug-in deployment fails, the log file is available in $ORACLE_HOME/cfgtoollogs/pluginca/*. Append these files when logging a support request for failure while deploying the plug-in. You can also use them to debug the problem.

## Important Details Regarding Plug-in Deployment

- You can import multiple versions of the same plug-in. The version to deploy can be selected from a list if you are using Cloud Control to deploy the plug-in, or can be specified on the command line if using EM CLI.

- Only one version can be deployed on the Oracle Management Service (OMS) at any given time. If a later version has been deployed previously, it cannot be downgraded to an earlier version.

- Updating a plug-in to a new version does not remove the content of the older plug-in(s) that were imported.

- The Management Agent can have the same or earlier version of the plug-in that is deployed on the OMS. A version later than the version on the OMS is not allowed on the Management Agent.

- The plug-in on OMS and the Management Agent can be updated independently of each other as long as the version on the OMS is the latest version.

- Available updates are visible on the Plug-ins page. They can be downloaded from the Enterprise Manager store or imported using EM CLI as described in Importing the Plug-in Archive.

# Adding a Target Instance

When the plug-in is deployed on OMS, it is ready to monitor target instances.

> **Note:**
>
> In the current Cloud Control release, deployment of a plug-in to a Management Agent that will monitor targets is no longer required. Instead, the plug-in for a specific target type is automatically deployed with the Management Agent that will monitor targets of that type.
>
> This is a significant change from previous releases, in which plug-ins had to be manually deployed to a Management Agent first. Then a target instance had to be added to the Management Agent manually.

You can add targets that the plug-in will monitor through Enterprise Manager Cloud Control by selecting **Add Targets** from the **Setup** menu. The process for adding targets - known in Cloud Control terminology as *target promotion* - varies depending on the option you choose.

You can also add a target instance using the EM CLI utility. Open a command prompt and run the following command:

```
emcli add_target
        -name="name"
        -type="type"
        -host="hostname"
        [-properties="pname1:pval1;pname2:pval2;..."]...
        [-separator=properties="sep_string"]
```

```
        [-subseparator=properties="subsep_string"]
        [-credentials="userpropname:username;pwdpropname:password;..."]
        [-input_file="parameter_tag:file_path"]
        [-display_name="display name"]
        [-groups="groupname1:grouptype1;groupname2:grouptype2;..."]...
        [-timezone_region="gmt offset"]
        [-monitor_mode="monitor mode"]
        [-instances="rac database instance target name1:target
type1;..."
]
```

For example:

```
emcli add_target
        -name="cluster_database"
        -type="rac_database"
        -host="myhost.us.example.com"
        -monitor_mode="1"
        -
properties="ServiceName:service.us.example.com;ClusterName:newdb_cluster
"
        -
instances="database_inst1:oracle_database;database_inst2:oracle_database
"
```

Use the `emcli help add_target` help command to see more options when adding the target instance.

If targets have been added before, they will be promoted and monitored by the plug-in after it is deployed.

# Updating Deployed Metadata Files Using the Metadata Registration Service (MRS)

As part of the plug-in development process, you will package your plug-in as an archive and deploy it to an Enterprise Manager Cloud Control installation to test it. However, you will likely not want to re-deploy the plug-in each time you make changes to various metadata files.

> **✎ Note:**
>
> You must update the metadata version each time you update a metadata file.

The Metadata Registration Service (MRS) allows you to upload one or more updated metadata files to the Oracle Management Service and Management Agents your plug-in has been deployed to. The updated metadata files will be registered with Enterprise Manager, and will overwrite the existing metadata with your updates.

> **✎ Note:**
>
> For target types and default collections, some additional steps are required for using MRS, see Target Types and Default Collections.

This service is invoked through the `emctl register oms metadata` command. The syntax is as follows:

```
emctl register oms metadata -service Metadata Service Id (-file metadata file
to register | -file_list file containing list of files to register)
         [-core | -pluginId Plugin Id] [-sysman_pwd "sysman password"]
```

For example, the following command registers changes to target type metadata file:

```
emctl register oms metadata -service targetType -file /staging/
demo_hostsample.xml -pluginId test.demo.xyz -sysman_pwd myempassword
```

Table 14-4 describes the usage of the command.

**Table 14-4    emctl Command Usage**

| Option | Required Y/N | Description |
| --- | --- | --- |
| `-service` | Y | Specify the type of metadata to register. Values are:<br>• `targetType`: Specify for target type metadata.<br>• `default_collection`: Specify for default collection metadata.<br>• `LiveSnapshotRegistration`: Specify for configuration metadata registration<br>• `CredStoreMetadata`: Specify for<br>• `jobTypes`: Specify for<br>• `report`: Specify for Report Metadata Registration<br>• `bipublisherreport`: Specify for BI Publisher report metadata.<br>• `discovery`: Specify for discovery metadata.<br>• `derivedAssocs`: Specify for associations metadata.<br>• `gccompliance`: Specify for compliance rules, compliance standards, and compliance framework metadata.<br>• `mpcui`: Specify for management user interface metadata. |
| `-file` | N | The path and file name for a single metadata file to upload and register. Either `-file` or `-file_list` can be included. |
| `-file_list` | N | The path and file name for a file containing a list of metadata file paths (one on each line). |
| `-core` | N | Not valid for plug-in development. |

**Table 14-4    (Cont.) emctl Command Usage**

| Option | Required Y/N | Description |
|---|---|---|
| `-pluginId` | N | The unique three-part identifier given to the deployed plug-in to update. See Defining the Plug-in ID for details. |
| `-sysman_pwd` | Y | The Enterprise Manager user password. |

## Target Types and Default Collections

For target types and default collections, some additional steps are required for using MRS if there are existing targets of this target type.

If you have an existing target and you want to update the metadata files during the plug-in development process, follow these steps:

1.  Register the new metadata files using the `emctl register oms metadata` command.

    ```
    emctl register oms metadata -service targetType -file full path/
    TargetTypeMetadata.xml -pluginId Plugin Id -sysman_pwd sysman

    emctl register oms metadata -service storeTargetType -file full path/
    TargetTypeMetadata.xml -pluginId Plugin Id -sysman_pwd sysman

    emctl register oms metadata-service default_collection -file full path/
    TargetTypeCollection.xml -pluginId Plugin Id -sysman_pwd sysman

    emctl register oms metadata-service systemStencil -file full path/
    TargetTypeStencil.xml -pluginId Plugin Id -sysman_pwd sysman
    ```

2.  Place the metadata XML files in the correct directories of the plug-in home directory (*PLUGIN_AGENT_HOME*) in the Management Agent as shown. The PLUGIN_AGENT_HOME directory is created when the plug-in is deployed to the Management Agent. The default location is *AGENT_BASE_DIR*/plugins.

    ```
    $PLUGIN_AGENT_HOME/metadata/
    $PLUGIN_AGENT_HOME/default_collection
    ```

3.  Restart the Management Agent.

    ```
    AGENT_HOME/agent/bin/emctl stop agent
    AGENT_HOME/agent/bin/emctl start agent
    ```

    In the preceding command, *AGENT_HOME* represents the Management Agent home directory.

# 15

# Defining Software Library Metadata

The chapter introduces Software Library framework, and describes how you can define and register metadata that is used by plug-in integrators, to extend a Software Library to include extensions, and Out-of-box entities. After registering the custom extensions, you can use interfaces like Software Library console, EM CLI, Action Script API, and so on, to create, manage, and access Software Library entities. Primarily, the chapter contains the following topics:

- Introduction to Software Library Framework
- Defining Software Library Metadata
- Organizing Software Library Metadata Files
- Adding the Software Library Metadata to Enterprise Manager
- Using Software Library Entities

> **✐ Note:**
>
> For conceptual information about any of the topics covered in this chapter, see *Oracle Enterprise Manager Cloud Control Extensibility Programmer's Guide*.

## Introduction to Software Library Framework

After defining and registering the new metadata, you can extend the Software Library framework to become aware of new types of software or scripts or configurations. You can use these entities in the existing or custom automation logic represented by a job type or deployment procedure.

As a plug-in developer, you will be able to define the following Software Library metadata:

- **Folder**

  Software Library allows you to organize the different user-defined or out-of-box entities into logical folders for efficient management.

  For more information, see Defining Folders.

- **Type and Subtype**

  A type and subtype together define the common traits of the entities it represents in terms of common and searchable metadata/configuration properties, their default values, file association requirements, etc. All entities in Software Library have an associated type and a subtype.

  For more information, see Defining Types and Defining Subtypes.

- **Entity**

Entities are the primary artifacts stored in Software Library. They are identified by the type/subtype they have been created with and the folder they are present in.

For more information, see Defining Entities.

# Defining Software Library Metadata

This section contains the following topics:

- Defining Folders
- Defining Types
- Defining Subtypes
- Defining Entities

> **Note:**
>
> To view an example which describes how to use the Software Library artifacts, follow these steps:
>
> 1. In Cloud Control, from **Setup** menu, select **Extensibility**, and click **Development Kit**.
>
> 2. On the Extensibility Development Kit page, in the Getting Started section, five samples are listed in the samples directory. The Sample Host Plugin 2 (Sample II) is the plug-in sample that covers information about Software Library artifacts.
>
> Once you have downloaded the EDK kit to your local system, select `oracle.samples.xsh2` sample available in `/samples/plugins/` directory. If you further drill down to `/samples/plugins/oracle.samples.xsh2/plugin_dist/oms/metadata` directory, the `swlib` folder is displayed which contains the necessary examples.

> **Note:**
>
> All the Software Library Metadata described in this section must be included in the XML file as described in Organizing Software Library Metadata Files. Following which, you must register them using the information available in Adding the Software Library Metadata to Enterprise Manager.

## Defining Folders

The following example describes how you can create two top level folders called `MPFolder1` and `MPFolder2`, with a subfolder named `Subfolder` under each of these folders:

```
<Folders>
    <Folder name="MPFolder1">
    </Folder>
    <Folder name="MPFolder2">
```

```
        </Folder>
        <Folder name="MPFolder1/subfolder">
        </Folder>
        <Folder name="MPFolder2/subfolder">
        </Folder>
</Folders>
```

After defining and registering the folders, when the plug-in is deployed, these folders will be displayed in Software Library console with a lock symbol, which means that the folders are Oracle-Owned, and can not be editted. You can customize them using the Create Like functionality available in Software Library. All the other folders that are created using Enterprise Manager console by users appear without the lock symbol, and can be editted by anyone who has been given the required accesses on the folder.

# Defining Types

The following example describes how to create two Type artifacts called `MPType1` and `MPType2`:

```
<Types>
<EntityType internalName="MPType1"/>
<EntityType internalName="MPType2"/>
</Types>
```

# Defining Subtypes

The following example describes how you can create a subtype `MPSubtype1` for the type `MPType1`:

```
<EntitySubtype internalName="MPSubtype1" type="MPType1">
      <EntityProperties filename="MPSubtype-entPropDict.xml"/>
       <GenericUISpecification>
         <Create>
           <ContentDescriptor>
             <DisplayName default="Describe"/>
             <Description default="Describe"/>
             <Content contentId="/WEB-INF/sdk/
core/regions/swlib/sdkcore-regions-swlib-describe-task-flow.xml#sdkcore-
regions-swlib-describe-task-flow" contentType="ADFRegion"/>
           </ContentDescriptor>
           <ContentDescriptor>
             <DisplayName default="Select Files"/>
             <Description default="Select Files"/>
             <Content contentId="/WEB-INF/sdk/core/regions/swlib/
sdkcore-regions-swlib-upload-task-flow.xml#sdkcore-regions-swlib-upload-
task-flow" contentType="ADFRegion"/>
           </ContentDescriptor>
           <ContentDescriptor>
             <DisplayName default="Review"/>
             <Description default="Review"/>
             <Content contentId="/WEB-INF/sdk/core/regions/swlib/
sdkcore-regions-swlib-review-task-flow.xml#sdkcore-regions-swlib-review-
task-flow" contentType="ADFRegion"/>
           </ContentDescriptor>
```

```
            </Create>
            <Edit>
              <ContentDescriptor>
                <DisplayName default="Describe"/>
                <Description default="Describe"/>
                <Content contentId="/WEB-INF/sdk/
core/regions/swlib/sdkcore-regions-swlib-describe-task-flow.xml#sdkcore-
regions-swlib-describe-task-flow" contentType="ADFRegion"/>
              </ContentDescriptor>
              <ContentDescriptor>
                <DisplayName default="Select Files"/>
                <Description default="Select Files"/>
                <Content contentId="/WEB-INF/sdk/core/regions/swlib/
sdkcore-regions-swlib-upload-task-flow.xml#sdkcore-regions-swlib-upload-
task-flow" contentType="ADFRegion"/>
              </ContentDescriptor>
            </Edit>
            <View>
              <ContentDescriptor>
                <DisplayName default="Describe"/>
                <Description default="Describe"/>
                <Content contentId="/WEB-INF/sdk/
core/regions/swlib/sdkcore-regions-swlib-describe-task-flow.xml#sdkcore-
regions-swlib-describe-task-flow" contentType="ADFRegion"/>
              </ContentDescriptor>
              <ContentDescriptor>
                <DisplayName default="Select Files"/>
                <Description default="Select Files"/>
                <Content contentId="/WEB-INF/sdk/core/regions/swlib/
sdkcore-regions-swlib-upload-task-flow.xml#sdkcore-regions-swlib-upload-
task-flow" contentType="ADFRegion"/>
              </ContentDescriptor>
            </View>
          </GenericUISpecification>
        </EntitySubtype>
```

Once a subtype defined is registered, entities of this subtype can be created (if creation is not blocked) using the Software Library console. To do so, from Enterprise menu, select **Provisioning and Patching ,** then click **Software Library.** From the Software Libray page, select a user-owned folder. From **Actions** menu, select **Create**, and then select **Type** and then the **SubType.** All the sub-types that belong to the type selected are displayed. For more information, see the following graphic file.
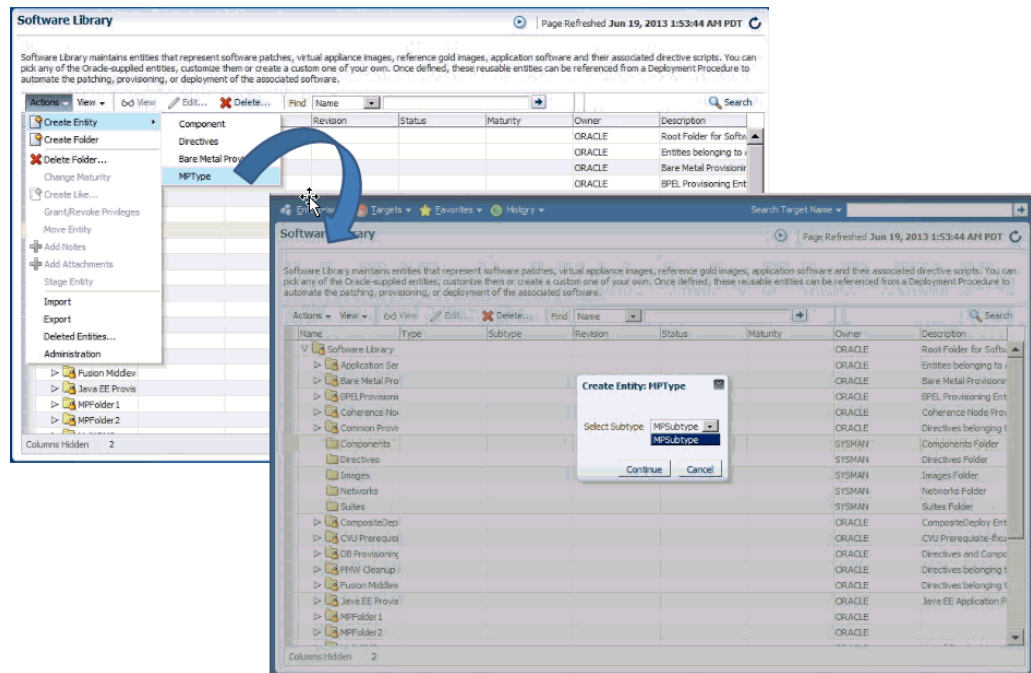
Ideally, you must see the subtypes you created as a part of this list, if not, there is an issue with registration. For more information on registering the sub-type, see Adding the Software Library Metadata to Enterprise Manager.

## Entity Properties File

Following is the entity properties file `MPSubtype-entPropDict.xml`:

```
<?xml version="1.0"?>
<Dictionary xmlns="http://www.oracle.com/sysman/emgc/Properties">
    <TypeDefinitions>
        <DictionaryItem refID="MPAttr1" purpose="PURPOSE_TopLevel">
            <PropType typeCode="TYPE_String">
                <SimpleType className="java.lang.String"
uiHintReadOnly="false" uiHintHidden="false" secret="false" guid="false">
                    <Constraints/>
                </SimpleType>
            </PropType>
        </DictionaryItem>

        <DictionaryItem refID="MPAttr2" purpose="PURPOSE_TopLevel">
            <PropType typeCode="TYPE_String">
                <SimpleType className="java.lang.String"
uiHintReadOnly="false" uiHintHidden="false" secret="false" guid="false">
                    <Constraints/>
                </SimpleType>
            </PropType>
        </DictionaryItem>

        <DictionaryItem refID="MPAttr3" purpose="PURPOSE_TopLevel">
            <PropType typeCode="TYPE_String">
```

```
            <SimpleType className="java.lang.String"
uiHintReadOnly="false" uiHintHidden="false" secret="false" guid="false">
                <Constraints/>
            </SimpleType>
        </PropType>
      </DictionaryItem>
    </TypeDefinitions>
    <DynamicTypes/>
</Dictionary>
```

To use the file, ensure that you make the following changes:

- `MPAttr1`, `MPAttr2`, and `MPAttr3` are the names of the entity properties, remove the properties that are not required, and change the names of the properties that you retain appropriately.

  Copy the `DictionaryItem` section from the above sample to create more entity properties. Note that, you can only update the name, you cannot change other aspects of the entity property.

- The entity properties for this sub-type are displayed as `Other Attributes` in Sotware Library console. Only, after the subtype metadata with entity property definitions are registered, while creating or editing entities, you can specify values for the entity properties. However, note that you can not define new entity properties.

## Defining Entities

The following example describes how you can create an entity of the type `MPType` and subtype `MPSubtype` in the folder `MPFolder1`:

```
<Entity name="MPTSEntity">
  <Type>MPType</Type>
  <Directory>/MPFolder1</Directory>
  <Subtype>MPSubtype</Subtype>
  <Fileset>
    <FileEntry>
      <path>payload.zip</path>
      <sourcePath>payloadSrc.zip</sourcePath>
    </FileEntry>
  </Fileset>
  <ExternalID>0.1</ExternalID>
</Entity>
```

**FileSet:** A file `payloadSrc.zip` should be present in the same directory as of this XML. The entity will have a file entry `payload.zip`.

**ExternalID :** A decimal floating point number of the form `xxxxxxxx.x`, which means, a maximum 8 digits before decimal point, and one digit after decimal point is allowed. For example, 0.6, 23.9, an so on. When registering metadata again, the external IDs of the entities are compared. Entities whose ExternalID matches the ExternalID of the latest revision currently registered will not get registered again.

For example:

- If the entity `MPTSEntity` in `MPFolder1` is registered with ExternalID 0.1, the an entity with revision 0.1 is created. If ExternalID is not specified, then the default value is 0.1.

- If XML is updated and registered again without changing ExternalID, then his entity will not be registered, and a warning will be logged. Entity will continue to have 0.1 revision.

- If the ExternalID is changed to 0.2, and registered again, then the registration will be applied, and a revision 0.2 is created in addition to the earlier registered 0.1 revision. Consequently, multiple increments might need to be performed to ExternalID for it to be registered during the development and testing phase. However, before the release, ensure that you reset it back to the correct value and track it correctly.

# Organizing Software Library Metadata Files

To organize the Software Library Metadata files, follow these steps:

1. Once you have the XMLs to define types, subtypes, folders, and entities ready, navigate to the following location:

   `$OMS_PLUGIN/metadata/swlib`

2. Create a directory for your functional area (for example, `functionalArea`), in the following location:

   `$OMS_PLUGIN/metadata/swlib/functionalArea`

3. Create a driving file (`swlib.xml`)

4. Edit `order.xml` file available at the following location: `$OMS_PLUGIN/meadata`, to add an entry for the newly created `functionalArea/swlib.xml` file:

Here is a sample example of how your directory structure should look:

```
swlib/
    order.xml
    functionalArea/swlib.xml
    functionalArea1/swlib.xml
```

Here is a sample example of the contents of `order.xml` file:

```
<order>
  <name>functionalArea/swlib.xml</name>
  <name>functionalArea1/swlib.xml</name>
</order>
```

> **Note:**
>
> For each functional area in the plug-in, you must add an entry in the `order.xml` file as described in the example.

# Adding the Software Library Metadata to Enterprise Manager

Adding the Software Library Metadata to Enterprise Manager is a two-step process as described in this section:

- Step 1: Validating Metadata XML
- Step 2: Adding Metadata XML to OPAR

## Step 1: Validating Metadata XML

For the purpose of testing, use emctl to register the metadata as follows:

```
emctl register oms metadata -service swlib -file <Metadata Instance file> -
pluginId <Plugin Id> [-sysman_pwd "sysman password"]
Where:
Metadata Instance file is the path to the folder containing order.xml. For
example, $OMS_PLUGIN/metadata/swlib.
```

> **Note:**
>
> If you have not added a Software Library location, then the `emctl register` command will not work, instead, you will see an error message as follows:
>
> ```
> EM-04040: Metadata operation is skipped. Reason: Software Library OMS
> shared storage is not configured, skipping metadata registration.
> Check /u01/inst/em/EMGC_OMS1/sysman/log/emctl.log for more details.
> ```

Also, if the metadata XML file is repeatedly registered for entities, then you must ensure that the external ID element is incremented each time. For example, if there are 11 entities defined in XML, four of which have same ExternalID, and seven have updated ExternalID, then you will see the following message:

```
Total 0 errors, 4 warnings. 7 entities imported.
Metadata registration successful
```

> **Note:**
>
> OMS must be restarted for the registration to take effect, when a metadata registered using emctl contains types or subtypes. However, OMS need not be restarted for the registration of types and subtypes, when the Software Library metadata defined in a plug-in contains types or subtype, and is deployed.
>
> Oracle recommends that you check the logs even if the registration is successful, as there may be some warnings. These warnings are mainly caused when the external ID is not modified, inturn causing entity registration to fail. To view the logs, navigate to the following location:
>
> ```
> $INSTANCE_HOME/sysman/log/emctl.log
> ```

## Step 2: Adding Metadata XML to OPAR

When a plug-in OPAR is deployed containing Software Library metadata organized in the way described above, Software Library metadata will be registered if Software Library is configured on the system. Software Library metadata XMLs are to be included in the OPAR like any other metadata. See Chapter 13, "Validating, Packaging, and Deploying the Plug-in" for more information.

Note 1: If Software Library is not configured at plug-in deployment time, the plug-in's Software Library metadata will get registered whenever Software Library is configured for the first time after the plugin deployment.

# Using Software Library Entities

Software Library entities created by the plug-in may represent a patch/script/ configuration or any other software relevant to the plug-in. To use these entities after they have been created using the options described in previous sections, consider one of the following approaches:

- Using Job Types
- Using EMCLI Verbs

> **✎ Note:**
>
> For information about how plug-in Graphical User Interface uses the Software Library search service, see Software Library Search Service .
>
> For information about how to use the Component step and Directive Step available in User Defined Deployment Procedure to automate a custom deployment activity, see *Oracle Enterprise Manager Lifecycle Management Administrator's Guide*

## Using Job Types

Software Library makes use of the following job types:

- `SwlibStageEntities` - Transfers files associated with a Software Library Entity to a destination host target.

- `SwlibUploadFiles` - Uploads files to be associated with a Software Library Entity to the appropriate Software Library storage location specified.

You can create your own job types, which inturn contain these jobtypes. For example, you can create a jobtype XML that containsthe `SwlibStageEntities` jobtype as follows:

```xml
<?xml version="1.0"?>
<jobType name="StageWrap" version="1.0" singleTarget="true" targetTypes="host"
editable="true">
    <credentials>
    <credential usage="destHostCreds" authTargetType="host"
defaultCredentialSet="HostCredsNormal">
    </credential>
    <credential usage="destNfsHostCreds" authTargetType="host"
defaultCredentialSet="HostCredsPriv">
    </credential>
    </credentials>
    <paramInfo>
        <paramSource sourceType="user" paramNames="stageLocation, entityURN"
                     required="true" evaluateAtSubmission="true" />
        <paramSource sourceType="user" paramNames="stageFileEntryPaths,
operMode, autoRetry" required="false"/>
        <paramSource sourceType="inline" paramNames="operMode">
           <sourceParam name="paramValues" value="mount"/>
           <sourceParam name="overwriteExistingFiles" value="yes"/>
        </paramSource>
    </paramInfo>
    <stepset ID="main" type="serial">
      <step ID="preStage" command="remoteOp">
        <credList>
          <cred usage="defaultHostCred" reference="destHostCreds"/>
        </credList>
       <paramList>
        <param name="remoteCommand">%job_default_shell%</param>
        <param name="args">ls, -R, %stageLocation%</param>
        <param name="targetName">%job_target_names%[1]</param>
        <param name="targetType">%job_target_types%[1]</param>
      </paramList>
      </step>
```

```
            <job ID="stage" type="SwlibStageEntities">
              <credList>
                <cred usage="destHostCreds" reference="destHostCreds"/>
                <cred usage="destNfsHostCreds" reference="destNfsHostCreds"/>
              </credList>
              <paramList>
                <param name="entityURN">%entityURN%</param>
                <param name="stageFileEntryPaths" valueOf="stageFileEntryPaths"/>
                <param name="stageLocation">%stageLocation%</param>
                <param name="operMode">%operMode%</param>
                <param name="autoRetry">%autoRetry%</param>
                <param name="overwriteExistingFiles">%overwriteExistingFiles%</param>
              </paramList>
              <targetList allTargets="true" />
            </job>
            <step ID="postStage" command="remoteOp">
              <credList>
                <cred usage="defaultHostCred" reference="destHostCreds"/>
              </credList>
             <paramList>
              <param name="remoteCommand">%job_default_shell%</param>
              <param name="args">ls, -R, %stageLocation%</param>
              <param name="targetName">%job_target_names%[1]</param>
              <param name="targetType">%job_target_types%[1]</param>
             </paramList>
             </step>
          </stepset>
        <displayInfo
                    useDefaultCreateUI="true"
                    showParams="true">

        <jobTypeDisplayInfo>
          <nlsValue>Stage Wrap</nlsValue>
        </jobTypeDisplayInfo>
          <parameterDisplayInfo name="stageLocation" showInResults="true"
    showInCreate="true">
              <parameterLabel>
                  <nlsValue>Stage Location</nlsValue>
              </parameterLabel>
              <parameterHint>
                  <nlsValue>Directory location on target where the files from the
    entity will be transferred.</nlsValue>              </parameterHint>
              <parameterTextBox lines="1" />
          </parameterDisplayInfo>
          <parameterDisplayInfo name="entityURN" showInResults="true"
    showInCreate="true">
              <parameterLabel>
                  <nlsValue>Entity URN</nlsValue>
              </parameterLabel>
              <parameterHint>
                  <nlsValue>Internal identifier of the entity</nlsValue>
              </parameterHint>
                <parameterTextBox lines="1" />
          </parameterDisplayInfo>
        </displayInfo>
    </jobType>
```

Jobs of jobtype `SwlibStageEntities` expect the following inputs:

- Stage Location: The directory path where the file of the entity will be transferred.

- Entity URN: This is the internal identifier of the entity, which can be obtained any one of the following methods:

  – In Enterprise Manager Cloud Control, from the **Enterprise** menu, select **Provisioning and Patching**, then click **Software Library.** On the Software Library home page, from **View** menu, select **Internal ID** to enable it. Copy and supply the Internal ID value available to the job as the value of this parameter.

  – You can use the emcli verb `list_swlib_entities` with the parameter-`show_entity_rev_id` to obtain the Internal ID. Copy and supply the Internal ID value available to the job as the value of this parameter.

# Using EMCLI Verbs

You can use the EMCLI verbs provided by Software Library to add Software Library storage location, create folders, create entities, upload files for entities, modify entities, and so on. For more information about EMCLI verbs, see Oracle Enterprise Manager Command Line Interface and the Oracle Enterprise Manager Lifecycle Management documentation available in the Oracle Help Center:

https://docs.oracle.com/en/enterprise-manager/

# 16

# Defining Credentials

As part of the target type definition, you can define the types of credentials specific to the plug-in target type. For example, you can define the username and password required by the plug-in to connect to a target instance to collect metric data, or to invoke a specific Enterprise Manager job.

The Enterprise Manager credential subsystem enables Enterprise Manager administrators to store credentials in a secure manner as preferences or operation credentials. The credentials can then be used to perform various system management activities such as real-time monitoring, patching, provisioning, and other target administrative operations.

In this release, the credential subsystem supports storing, accessing, and modifying of fixed number user name/password based credentials as preferred credentials, which other Enterprise Manager subsystems access to build automation solutions. The credential subsystem also supports sudo/powerbroker based impersonation support.

This chapter covers the following:

- Introduction to Security Concepts
- Defining Credential Metadata

## Introduction to Security Concepts

The following sections describe the concepts associated with credential service integration:

- Credential Types

  Credential type is the type of authentication supported by a target type. Various authentication schemes are supported, including native agent authentication and SSH. Fore more information, see Understanding Credential Types.

- Named Credentials

  A named credential contains a users' authentication information on a system and can be a user name and password, a public key-private key pair, or an X509v3 certificate. For more information, see About Named Credentials.

- Authentication Target Type

  An authenticating target type is the target type that a credential can authenticate against. For more information, see Authenticating Target Types.

- Credential Sets

  A credential set is a placeholder for a credential and can be used to decouple credentials from the system that uses a credential. For more information, see Overview of Credential Sets

- Credential Store

The credential store is a logical store for all the named credentials of an Enterprise Manager administrator in the Enterprise Manager. For more information, see Using the Credential Store

- Credential Reference

  The credential reference refers to a credential. For more information, see About the Credential Reference

## Understanding Credential Types

Credential type is the type of authentication supported by a target type. For example, a host can support a user name and password based authentication, public key authentication, or kerberos authentication. Various authentication schemes are supported, including native agent authentication and SSH.

The native agent authentication scheme employs a user name and password structure, while the SSH key authentication scheme uses a user name/private key/public key structure.

## About Named Credentials

A named credential contains a users' authentication information on a system. A named credential can be a user name and password, a public key-private key pair, or an X509v3 certificate. An Enterprise Manager administrator can store these credentials as named entities in Enterprise Manager to use when performing operations such as running jobs, patching, and other system management tasks. For example, you can store the user name and password that you want to use for patching as `MyPatchingCreds`. You can then later submit a patching job that uses `MyPatchingCreds` to patch the production databases.

Named credentials can be created for the credential types in Enterprise Manager 12*c*. The most commonly used credential types for host and database target types are described in the following sections.

For more information about named credentials, see the Configuring and Using Target Credentials section in the *Oracle Enterprise Manager Cloud Control Security Guide*.

## Authenticating Target Types

The authenticating target type is the target type that a credential can authenticate against. For example, a SQLScript job has the host credential `DBHostCreds` that is used to authenticate against the database host. Therefore, the target type for `DBHostCreds` is Database Instance and the authenticating target type is Host.

## Overview of Credential Sets

A credential set is a placeholder for a credential. Credential sets can be used to decouple credentials from a system that uses a credential. For example, a patching job can be submitted to use the credential set "Normal Host Credentials" while being executed.

The "Normal Host Credentials" credential set can also be set to the actual named credential. The credential set to named credential mapping for the target can be changed without editing the system that uses the credential.

## Using the Credential Store

The credential store is a logical store for all the named credentials of an Enterprise Manager administrator in the Enterprise Manager. The Enterprise Manager administrator's user name has a logical private credential store. Individual credentials can be identified by credential names. Enterprise Manager administrators can add, edit, and delete named credentials in the credential store.

## About the Credential Reference

The credential reference is a way to refer to a credential. There are three ways credentials can be referenced:

- Credential Name

  The credential is referenced using the name of the credential in the credential store.

- Credential Set

  The credential is referenced using the credential set name and the target name. The lookup retrieves the credential associated with the credential set name and target name.

- Direct

  The credential is specified by providing the values of the attributes. This reference does not refer to a credential in the credential store.

# Defining Credential Metadata

Credential metadata is defined within the target type metadata file. See Creating Target Metadata Files for more information about this file.

All credential metadata for a target type is defined within the `CredentialInfo` element. This element in turn contains the following subelements:

- A `CredentialType` element that defines the type of credentials to be used to access target instances
- A `CredentialSet` element that instantiates an instance of CredentialType

The following example defines username and password the credentials required to authenticate with hosts running instances of the target.

**Example: Credential Metadata**

```
<TargetMetadata>

...
 <CredentialInfo>
 <!-- The types of credentials: target host username/password -->
  <CredentialType NAME="HostCreds">
   <Display>
    <Label NLSID="CREDS_HOST_HOSTCREDS">Host Credentials</Label>
   </Display>
   <CredentialTypeColumn NAME="HostUserName" IS_KEY="TRUE">
   <Display>
    <Label NLSID="CREDS_HOST_USERNAME">UserName</Label>
```

```
     </Display>
    </CredentialTypeColumn>
    <CredentialTypeColumn NAME="HostPassword">
     <Display>
      <Label NLSID="CREDS_HOST_Password">Password</Label>
     </Display>
    </CredentialTypeColumn>
 </CredentialType>
<!-- The CredentialSet that creates an instance of CredentialType -->
 <CredentialSet NAME="HostCredsNormal" CREDENTIAL_TYPE="HostCreds"
  USAGE="PREFERRED_CRED">
    <Display>
     <Label NLSID="CREDS_HOST_HOSTCREDS_NORMAL">Normal Host Credentials</Label>
    </Display>
    <CredentialSetColumn TYPE_COLUMN="HostUserName" SET_COLUMN="username">
     <Display>
      <Label NLSID="CREDS_NORMAL_USER">Normal Username</Label>
     </Display>
    </CredentialSetColumn>
    <CredentialSetColumn TYPE_COLUMN="HostPassword" SET_COLUMN="password">
     <Display>
      <Label NLSID="CREDS_NORMAL_PASSWORD">Normal Password</Label>
     </Display>
    </CredentialSetColumn>
 </CredentialSet>
<CredentialInfo>
...
</TargetMetadata>
```

# Overview of Credential Elements

The key elements that define credentials are described in the following table:

**Table 16-1    Key elements in a plugin.xml file**

| Element | Required (Y/N) | Description |
| --- | --- | --- |
| CredentialInfo | Y | The root element for the credentials definition. Contains `CredentialType` and `CredentialSet` elements. |
| CredentialType | Y | Contains one or more `CredentialTypeColumn` elements, each defining a credential such as "TargetUsername" or "TargetPassword". Used to access target instances. |

**Table 16-1    (Cont.) Key elements in a plugin.xml file**

| Element | Required (Y/N) | Description |
| --- | --- | --- |
| CredentialSet | Y | Instatiates an instance of the credential set defined in `CredentialType`. It includes the following attributes:<br><br>• CREDENTIAL_TYPE<br>  Identifies the CredentialType from which this CredentialSet is created.<br>• USAGE<br>  Values are MONITORING (default), which is used to directly connect to the target, PREFERRED_CRED, which is the user's preferred credentials, or SYSTEM, which is used by specialized applications such as patching or cloning. |
| CredentialSetColumn | Y | A subelement of CredentialType. Defines a single credential and maps that credential to its corresponding column in the CredentialType. It includes the following attributes:<br><br>• `TYPE_COLUMN`<br>  Specifies the CredentialTypeColumn that this CredentialSetColumn maps to.<br>• `SET_COLUM`<br>  Identifies the column definition in the CredentialSet. |

# 17

# Defining a Chargeback Entity Type

Chargeback provides a way to meter and charge for resource use, where a resource is an entity known to Enterprise Manager. Typically, these entities are managed entities of type host, database, and WebLogic Server, for which specific metrics can be collected and charged. An administrator assigns rates and other usage factors to these metrics so they become charge items in something called a charge plan that is assigned to a target instance. A daily Chargeback job collects the metrics and calculates charges against resource use.

Although Enterprise Manager recognizes hundreds of entity types, relatively few are enabled for Chargeback out-of-box. As a plug-in developer you can use the extensibility feature to enable Chargeback on entity types defined within the plug-in, by leveraging extensibility Metadata Services (MDS). To take advantage of MDS, you create an XML file that models a new Chargeback entity type on an Enterprise Manager managed entity (ME) and defines charge items based on Enterprise Manager metrics and configurations.

This chapter describes how to define a new entity type to be added to Chargeback. The chapter contains the following sections:

- Chargeback Extensibility Toolkits
- Steps to Develop and Test New Chargeback Entity Type
- The Chargeback Model
- Sample Chargeback MDS XML File
- Registering the Chargeback MDS
- Testing the Entity Type Plug-in

## Chargeback Extensibility Toolkits

Besides the internal implementation of the Chargeback extensibility framework, the following toolkits are available to plug-in developers:

- Chargeback metadata service (MDS) registration XML schema

  ```
  emSDK\emMrsXsds\oracle\sysman\emSDK\chargeback\ChargebackMetadata.xsd
  ```

- Chargeback callback signature and built-in implementation (built-in callback enables the plug-in developer to register the callback in the Chargeback metadata file for the new entity type)

  Included in the pl/sql package `GC$CHARGEBACK`

  The pl/sql callback has the following signature:

  ```
  PROCEDURE add_entity_callback_name(
  p_em_entity_guid IN RAW,
  p_usage_mode_name IN VARCHAR2 DEFAULT NULL,
  p_entity OUT GC$CBA_ENTITY);
  ```

Using these tools, you can define Chargeback metadata to provide Chargeback support for a new entity type of a managed Enterprise Manager target type and register the built-in callback so that entity instances can be added using the Chargeback user interface and incorporated in the daily job schedule.

# Steps to Develop and Test New Chargeback Entity Type

The basic flow to develop and test a new Chargeback entity type involves the following tasks:

1. Define the Chargeback metadata for the entity and charge items. Include the appropriate callbacks to interact with the Chargeback user interface.

2. Register the Chargeback metadata file. An alternate course of action is to include the XML file in a plug-in and deploy the plug-in.

3. Create a charge plan in Chargeback that includes the charge items defined in the metadata.

4. Add the new entity type to Chargeback using the Add Entities wizard.

5. Assign the charge plan you created to the added entity type.

6. Trigger the daily data collection job to populate the tables.

7. Run reports in Chargeback on the new entity type to view charges and metrics.

# The Chargeback Model

This section describes the basic concepts as they relate to the primary elements within the Chargeback metadata file.

**Enterprise Manager Entity Type**

The Enterprise Manager entity type serves as a model for the Chargeback entity type to be defined. Typically it is an Enterprise Manager target type recognized as a manageable entity (ME).

**Chargeback Entity Type**

The Chargeback entity type is modeled after its container Enterprise Manager entity type. The Chargeback entity type is characterized by its usage mode; that is, how it is to be metered. Usage mode can involve parent-child relationships with other Chargeback entity types. A Chargeback entity type that is metered directly typically has a set of charge items defined.

More than one Chargeback entity type can be modeled after an Enterprise Manager entity type, but only one can be active in a given release.

**Usage Mode**

Usage mode defines how the Chargeback entity type is to be metered: directly or through its member entity types. It can also indicate which charge items to meter. A Chargeback entity type can have different usage modes to suit different situations, but only one can be the default.

**Charge Template**

The charge template indicates the charge items through which to meter the Chargeback entity type. One charge template serves for each usage mode related

to direct metering. A composite or parent Chargeback entity type does not require a charge template as they have no direct charge items.

**Charge Item**

A `ChargeItem` element in the Chargeback metadata file is akin to a charge item type definition. It defines the type of data to be collected and how.

A charge item can be based on an ME configuration, metric, or property. A charge item can also be a fixed amount.

**Entity Callback**

There are two entity instance level callbacks that can be registered for each Chargeback entity type.

- Add entity callback–called by the Chargeback Add Entities wizard when the administrator selects an entity to add to Chargeback. There are two built-in implementations, intended for plug-in developers:

```
gc$chargeback.add_em_entity_cb(
    p_em_entity_guid IN RAW,
    p_usage_mode_name IN VARCHAR2 DEFAULT NULL,
    p_entity OUT GC$CBA_ENTITY);

gc$chargeback.add_em_entity_and_members_cb(
    p_em_entity_guid IN RAW,
    p_usage_mode_name IN VARCHAR2 DEFAULT NULL,
    p_entity_arr OUT GC$CBA_ENTITY_ARR);
```

- Add member callback–called both by the Chargeback Add Entities wizard when the administrator selects an entity to add to Chargeback, and by the Chargeback data collection job to discover members that may have been added to the composite entity. There is one built-in implementation, also intended for plug-in developers:

```
gc$chargeback.add_entity_members_cb(
    p_em_entity_guid IN RAW,
    p_usage_mode_name IN VARCHAR2 DEFAULT NULL,
    p_entity_arr OUT GC$CBA_ENTITY_ARR);
```

Both `gc$chargeback.add_em_entity_and_members_cb` and `gc$chargeback.add_entity_members_cb` are implemented based on a `chargeback_parent` association.

The built-in callbacks do not apply automatically to the Chargeback entity type. The Chargeback entity instance will not be visible in the Add Entities wizard unless you register the applicable callback in the Chargeback metadata file. You cannot override these callbacks.

# Sample Chargeback MDS XML File

The sample Chargeback metadata file complements the Host Sample included in the EDK. It proposes to add Chargeback support for the Host Sample to include a fixed base charge for each host instance in addition to a rate charged for each gigabyte of storage and memory used on the host instance.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ChargebackMetadata name="demo_hostsample_cba" description="Chargeback
```

```
meta data for demo_hostsample" version="1.0"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <ChargebackEntityType name="demo_hostsample" displayLabel="Demo
Plugin Sample Host" labelNlsId="name" description="Chargeback entity
type for sample target type demo_hostsample"
                        descriptionNlsId=""
mappingEmEntityType="demo_hostsample"

resourceBundle="oracle.sysman.eml.rsc.chargeback.demo_hostsample_cbaMsg"
 defaultUsageMode="metered">
    <ChargeItems>
      <!-- fixed item -->
      <ChargeItem name="BaseCharge" displayLabel="Base Charge"
labelNlsId="em_ct_base_chg" type="fixed"
                  dataType="number" category="instance"
aggregationType="sum" description="Base charge for a demo_hostsample
instance">
        <QueryExpression type="internal"></QueryExpression>
      </ChargeItem>
          <!-- config -->
        <!-- usage -->
      <ChargeItem name="memUsageMB" displayLabel="Memory Usage"
labelNlsId="em_ct_mem_usg" type="metric"
                  dataType="number" unit="GB" unitNlsId="em_ct_u_GB"
category="memory"
                  aggregationType="avg" description="Memory used (GB)">
        <QueryExpression type="internal" valueColumn="value/(1024*1024)"
                         emEntityGuidColumn="entity_guid"
                         metricGroup="MemoryPerf"
metricColumn="MemTotal"
                         utcCollectionDateColumn="collection_time_utc">
        </QueryExpression>
      </ChargeItem>
      <ChargeItem name="diskUsageGB" displayLabel="Disk Space Usage"
labelNlsId="em_ct_disk_usg" type="metric"
                  dataType="number" unit="GB" unitNlsId="em_ct_u_GB"
category="storage"
                  aggregationType="avg" description="Disk storage used
(GB)">
        <QueryExpression type="internal" valueColumn="value/(1024*1024)"
                         emEntityGuidColumn="entity_guid"
                         metricGroup="FilesystemPerf"
metricColumn="UsedKB"
                         utcCollectionDateColumn="collection_time_utc">
        </QueryExpression>
      </ChargeItem>
    </ChargeItems>
    <ChargeTemplates>
      <ChargeTemplate name="demo_hostsampleMetered"
displayLabel="Charge template for metering demo_hostsample"
                      labelNlsId="http_dedicated"
description="Charegeback charge template for metering Http
demo_hostsample"
                      descriptionNlsId="demo_hostsample_desc"
```

```
usageMode="metered" isMaster="yes"></ChargeTemplate>
    </ChargeTemplates>
    <UsageModes>
      <UsageMode name="metered" displayLabel="Metered"
labelNlsId="metered" type="dedicated" isChargeable="yes"
                  isNavigational="no"
defaultChargeTemplate="demo_hostsampleMetered" description="The usage
is metered in demo_hostsample level."></UsageMode>
    </UsageModes>
  </ChargebackEntityType>
  <EntityCallbacks>
    <EntityCallback callbackProc="GC$CHARGEBACK.add_em_entity_cb"
type="addEmEntity" entityType="demo_hostsample"/>
  </EntityCallbacks>
</ChargebackMetadata>
```

Table 17-1 lists the key elements in the Chargeback MDS XML file used to define a new Chargeback entity type. See the ChargebackMetadata.xsd for the complete set of all elements and attributes available.

**Table 17-1    Key Elements for Defining a New Chargeback Entity Type**

| Element | Attribute | Description |
| --- | --- | --- |
| `ChargebackMetadata` | `name` | (Required) Unique name of the Chargeback entity set. |
| `ChargebackEntityType` | `name` | (Required) Unique name of the Chargeback entity type. |
| NA | `displayLabel` | (Required) Display label of the Chargeback entity type. |
| NA | `mappingEmEntityType` | Enterprise Manager manageable entity type of the Chargeback entity type, if applicable. |
| NA | `defaultUsageMode` | Default usage mode that applies to the Chargeback entity type. The entity type can have multiple usage modes, but only one can be the default. |
| `ChargeItem` | `name` | (Required) Unique name of the charge item. |
| NA | `displayLabel` | (Required) Display label of the charge item. |
| NA | `type` | (Required) The charge item's data source. Can be: `metric`, `config`, `property`, or `fixed`. |
| NA | `dataType` | (Required) Charge item data type. Can be: `string` or `number`. |
| NA | `aggregationType` | How to aggregate data collected hourly into a daily total. Acceptable values are `sum` (total the hourly numbers) or `avg` (take the average of the hourly numbers). Default is `avg`. |
| NA | `isChargeable` | Stipulates that the administrator can set a rate directly on the charge item when creating a charge plan. Default is `yes`. |

**Table 17-1    (Cont.) Key Elements for Defining a New Chargeback Entity Type**

| Element | Attribute | Description |
| --- | --- | --- |
| NA | canBeChargeSet Cond | Stipulates that the set of charge plan rates defined for a charge plan configuration apply to the Chargeback entity instance only if the instance has the same value as the plan conditional value for the charge item. Default is `no`. |
| NA | canBeChargeRat eCond | Stipulates that the administrator can set a conditional rate on the charge item such that it is applicable only for a Chargeback entity with the specific value on the item. Default is `yes`. |
| NA | category | Logical resource category of the charge item. Can be: `activity`, `cpu`, `instance` , `memory`, `storage`, `network`, `service`, `software`, `uptime`, or `unclassified`. Default is `unclassified`. |
| Key | name | (Required) Name of the charge item key column. |
| NA | displayLabel | (Required) Display label of the charge item key. |
| QueryExpression | type | (Required) Type of query expression. Defines how to collect the charge item data. For purposes of this discussion, `internal` is the expected value. This means that the metric, config, or property of the ME type is pulled into Chargeback based on the metric group and column names, the configuration view and column names or the property name, respectively. |
| NA | entityNameColu mn<br>emEntityTypeCo lumn<br>emEntityNameCo lumn<br>emEntityGuidCo lumn<br>utcCollectionD ateColumn<br>collectionDate Column<br>keyColumn | These attributes are common to all charge items. They define the data to be extracted from the Enterprise Manager repository for the entity type. |
| NA | metricGroup<br>metricColumn | These attributes are specific to charge items based on metrics. |
| NA | viewName<br>valueColumn | These attributes are specific to charge items based on configurations. |
| NA | propertyName<br>valueColumn | These attributes are specific to charge items based on target properties. |
| ChargeTemplate | name | (Required) Unique name of the charge template to used for the Chargeback entity type. |

**Table 17-1    (Cont.) Key Elements for Defining a New Chargeback Entity Type**

| Element | Attribute | Description |
|---|---|---|
| NA | displayLabel | (Required) The charge template display label. |
| NA | usageMode | (Required) Usage mode name associated with the charge template. |
| NA | isMaster | Stipulates that the charge template is the master template for the Chargeback entity type. |
| | | A master charge template contains all the charge items defined for the Chargeback entity type so there is no need to include an item list as part of the `chargeTemplate` element. In the absence of a master charge template, however, the element must include the list of charge items in the Chargeback metadata file. |
| | | Acceptable values are `yes` or `no`. Default is `no`. |
| UsageMode | name | (Required) Unique name of the Chargeback entity type's usage mode. |
| NA | displayLabel | (Required) Usage mode display label. |
| NA | type | Usage mode type. Acceptable values are `dedicated` (where the entity type is metered by instance, for example) or `shared` (where the entity type is metered by service, for example). Default is `dedicated`. |
| NA | defaultChargeTemplate | Charge template in the Chargeback entity type that can serve as the default for the current usage mode. |
| NA | isChargeable | Stipulates that an entity using this usage mode can be chargeable. |
| | | Acceptable values are `yes` or `no`. Default is `yes`. |
| NA | isNavigational | Stipulates that the entity using the this usage mode is navigational; that is, it has children. |
| | | Acceptable values are `yes` or `no`. Default is `no`. |
| EntityCallback | entityType | (Required) Name of the entity type to which it applies. |
| NA | type | The type of entity callback. Acceptable values are `addEmEntity` or `addMembers`. |
| NA | callbackProc | (Required) Identifies the entity callback procedure. |

**About NLS IDs**

NLS IDs are mostly optional in the XML file. If not defined explicitly in the file, they are generated internally based on a formula. For example:

- The entity type display name nls id becomes e_entity_type_name

- The entity type description nls id becomes e_entity_type_name_desc

- Similarly, usage mode display name nls id becomes u_usage_mode_name

- The usage mode description name nls id becomes u_usage_mode_name_desc

This autogeneration satisfies the requirements of the ResourceBundle file.

# Registering the Chargeback MDS

Manually run the registration service to register the plug-in by executing a command similar to the following:

```
emctl register oms metadata
  -service chargeback -core -sysman_pwd <sysmanPWD>
  -file <directory> demo_hostsample_cba.xml
```

Where `directory` is the location of the Chargeback metadata file. Upon successful registration, proceed with testing.

As an alternative to registering the plug-in, you can include the XML file in the plug-in and deploy the plug-in. The XML file appears in the `oms/metadata/chargeback` folder in the Oracle Plug-in Archive (OPAR) file. Administrators as well as plug-in developers can use this method.

# Testing the Entity Type Plug-in

Having defined and registered the plug-in, proceed with the following tasks to ensure proper setup. Tasks are outlined here. For details, see the Chargeback Administration chapter in the Cloud Administration Guide or use the Chargeback online help. All tasks assume that you are logged in to Enterprise Manager Cloud Control and working in Chargeback (select **Chargeback** from the **Enterprise** menu).

**Create a Charge Plan**

Create a charge plan for the Demo Plugin Sample Host that includes the charge items defined in the Chargeback metadata file.

1. On the **Charge Plans** tab, select **Plan** on the **Create** menu.

2. Name the plan (Demo Host Plan), then click **Add** to select the Demo Plugin Sample Host entity type.

3. Click **Add Item** and select the Base Charge item defined in the Chargeback metadata file. Click **OK**.

   Repeat for the other items defined in the Chargeback metadata file (Disk Space Usage and Memory Usage).

4. Set rates for the three charge items. Base Charge is a flat rate per period. The other two are per GB/period charges.

5. Click **Save** to complete plan creation.

**Add an Entity of the New Type**

Add an entity of the new type to Chargeback to track charge and metering data.

1. On the **Entities** tab, click the **Add Entities** button.

2. In the wizard, click **Add**.

3. Search for entities of type Demo Plugin Sample Host and select one to add to Chargeback. Leave the default usage mode (Metered). Click **Next**.

4. Select the row of the entity you just added and click the **Assign Plan** button.

5. Select the plan (Sample Host Plan) in the list and click **OK**. Click **Next**.

6. Review your selections and click **Submit**. Chargeback confirms that the entity was added. The entity appears in the table together with its charge plan assignment.

**Review Chargeback Data**

Chargeback data collection occurs on a 24-hour cycle, but you can cause it to happen on-demand. On the **Entities** tab, select **On-demand data collection** from the **Action** menu. The tables are updated to reflect the most recent data.

View charge and metering data for the Demo Plugin Sample Host entity type.

1. Select the **Reports** tab in Chargeback.

2. Search for Demo Plugin Sample Host under **Entities**.

3. Select a metric to view.

4. Click the **View Report** button to recalculate based on your selections. The summary graphs redraw to display charge percentages.

5. View details in the lower pane. Reconfigure the display by changing the options.

# 18

# Monitoring Using Web Services and JMX

You can extend Enterprise Manager to monitor Web services and JMX-instrumented applications for critical events, performance problems, error conditions, and statistics.

Enterprise Manager's ability to monitor WSDL and JMX-enabled targets enables you to consolidate monitoring and management operations. When added to the Enterprise Manager framework, Enterprise Manager functionality, such as notifications, jobs, and reporting, is automatically extended to these targets.

This chapter contains the following topics:

- Overview
- Monitoring Using Web Services in Enterprise Manager
- Monitoring Using WS-Management in Enterprise Manager
- Monitoring a Standalone JMX-instrumented Java Application or JVM Target
- Monitoring JMX Applications Deployed on Oracle WebLogic Application Servers
- Adding a Target to a Management Agent
- Monitoring Credential Setup
- Viewing Monitored Metrics
- Creating JMX Metric Extensions
- Surfacing Metrics from a Standalone JVM or Oracle Coherence
- Monitoring Using RESTful Services

## Overview

Using Enterprise Manager to monitor targets that expose a Web services management interface, JMX-instrumented applications and servers, and standalone Java Virtual Machine (JVM) targets entails defining a new target type via metadata plug-ins.

Creating a metadata plug-in consists of four basic steps:

1. Generate the target metadata and default collection files to be added to the plug-in.

2. Create an Oracle Plug-in Archive containing the target definition files for one or more plug-ins. A single archive may contain more than one plug-in.

3. Import the plug-in into Enterprise Manager.

4. Deploy the plug-in to the appropriate Management Agents.

   For more information about each of these steps, see Validating, Packaging, and Deploying the Plug-in .

Procedural information for the monitoring targets can be found in the following sections:

- **Monitoring Using Web Services in Enterprise Manager** discusses software components exposing an external interface that communicate across a network using a standard messaging protocol.

- **Monitoring a Standalone JMX-instrumented Java Application or JVM Target** discusses standalone Java applications running on J2SE5.0 or higher that are instrumented using JMX MBeans.

- **Monitoring JMX Applications Deployed on Oracle WebLogic Application Servers** discusses JMX applications running on Oracle WebLogic Application Servers 9.x or above.

Monitoring a Standalone JMX-instrumented Java Application or JVM Target explains how to generate metadata and default collection files for your custom JMX-enabled application by guiding you through the MBeans for which you are interested in collecting data, and helping you define the MBeans as metrics in Enterprise Manager. Even if your standalone Java application is not instrumented through JMX, you can still monitor the JVMs it is running on by directly creating the built-in JVM target instances as defined in Configuring a Standalone Java Application or JVM Target.

After the metadata and default collection files are created, you can follow the normal metadata plug-in mechanism to deploy your plug-in and create target instances of your Java application target type.

# Monitoring Using Web Services in Enterprise Manager

Web services are loosely coupled software components that expose an external interface via the Web Service Definition Language (WSDL). These components communicate across a network using a standard messaging protocol called Simple Object Access Protocol (SOAP). The Management Agent's Web service Fetchlet (with ID WSF) supports SOAP communication.

> **Note:**
>
> For more information about the Web services standard, see the World Wide Web Consortium (W3C) website:
>
> http://www.w3.org

**Prerequisites**

- Management Agent version 12.1.0.0.0 or later installed on that host.
- Oracle Management Server (OMS) version 12.1.0.0.0 or later with which the Management Agent communicates.

## Creating Metadata and Default Collection Files

Defining a target type to be monitored through a Web services interface includes creating the requisite target definition files, which are required to collect metrics from resources that support the WSDL interface:

- Target Metadata
- Default Collection

Enterprise Manager provides an easy-to-use Web services command-line tool that simplifies creating plug-ins by automatically generating these requisite files. Information retrieval is achieved through the Web services fetchlet that is integrated with the Management Agent.

The command-line tool works by parsing a specified WSDL file for all operations, and enables you to select one or more operations to be invoked. If multiple port types are specified in the WSDL file, the tool prompts you to select one of them. Operations are listed along with their parameters. A Web service operation can be one of four types:

- One Way
- Request Response
- Solicit Response
- Notification

The Request Response operation type is particularly useful: The selected operation could have primitive or complex parameters and results. The result of Web service invocation is displayed in a table (the tool prompts you to provide labels for the table columns). You can also filter result attributes by specifying an Xpath expression (see the `RowType` property in the generated target metadata, CalculatorService Target Metadata File example). Filter attributes can be useful for complex return types from which only few attributes are interesting.

The Web services command-line tool supports Web services with the following binding and encoding styles:

- DOC/literal
- DOC/Wrapped
- RPC/encoded

## Web Services CLI Command-line Tool Syntax

The Web services CLI command-line tool syntax is as follows:

```
emctl wscli [-metadata | -help] [-options]
```

The command accepts the following options:

- -wsdl=*file* | *URL*: WSDL file or URL (mandatory)
- -username=*user ID*: user name if the WSDL is protected

The command-line tool requires a WSDL file name or URL to locate the WSDL for a Web service. For example, for a Calculator service Web service, a WSDL URL would be as follows:

```
http://localhost:44861/CalWS/CalculatorPort?WSDL
```

The command tool script requires access to the Enterprise Manager home directory (EM_HOME) to run. The tool defaults to ORACLE_HOME (ensure this environment variable is set properly before using this tool).

The tool parses specified WSDL for all the port types and binding (supported protocols such as HTTP get/post, SOAP) to list all the operations. If there are multiple port types in WSDL, you will first be prompted to choose a port type.

# Web Services Command-line Tool Security

The command-line tool generates metadata required by Enterprise Manager for target monitoring purposes through the WSDL file. When you run this tool, you only require read permission on the WSDL file or URL and permission to save generated files to the appropriate directory.

# Generating the Files

The following example shows a sample WSDL file passed to the command-line tool to generate the target metadata and collection files.

**Example: Sample WSDL File CalculatorService.wsdl**

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
Oracle JAX-WS 2.1.5. -->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://tests.jaxws.oracle.com/"
xmlns:ns0="http://www.oracle.com/jaxws/tests"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" name="CalculatorService"
targetNamespace="http://tests.jaxws.oracle.com/">
    <wsdl:types>
        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://www.oracle.com/jaxws/tests/types">
            <xs:complexType name="calculatorFaultInfo">
                <xs:sequence>
                    <xs:element name="number" type="xs:int"/>
                    <xs:element name="reason" type="xs:string" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <xs:schema xmlns:ns1="http://www.oracle.com/jaxws/tests/types"
xmlns:tns="http://www.oracle.com/jaxws/tests" xmlns:xs="http://www.w3.org/2001/
XMLSchema" version="1.0" targetNamespace="http://www.oracle.com/jaxws/tests">
            <xs:import namespace="http://www.oracle.com/jaxws/tests/types"/>
            <xs:element name="CalculatorException" nillable="true"
type="tns:CalculatorException"/>
            <xs:element name="CalculatorWrapperException" nillable="true"
type="ns1:calculatorFaultInfo"/>
            <xs:complexType name="CalculatorException">
                <xs:sequence>
                    <xs:element name="Message" type="xs:string"/>
                    <xs:element name="Number" type="xs:int"/>
                    <xs:element name="Reason" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" xmlns:tns="http://tests.jaxws.oracle.com/"
targetNamespace="http://tests.jaxws.oracle.com/">
            <xsd:complexType name="add">
                <xsd:sequence>
                    <xsd:element name="arg0" type="xsd:int"/>
                    <xsd:element name="arg1" type="xsd:int"/>
                </xsd:sequence>
```

```
                    </xsd:complexType>
                    <xsd:element name="add" type="tns:add"/>
                    <xsd:complexType name="addResponse">
                        <xsd:sequence>
                            <xsd:element name="return" type="xsd:int"/>
                        </xsd:sequence>
                    </xsd:complexType>
                    <xsd:element name="addResponse" type="tns:addResponse"/>
                    <xsd:complexType name="square">
                        <xsd:sequence>
                            <xsd:element name="arg0" type="xsd:int"/>
                        </xsd:sequence>
                    </xsd:complexType>
                    <xsd:element name="square" type="tns:square"/>
                    <xsd:complexType name="squareResponse">
                        <xsd:sequence>
                            <xsd:element name="arg0" type="xsd:int"/>
                        </xsd:sequence>
                    </xsd:complexType>
                    <xsd:element name="squareResponse" type="tns:squareResponse"/>
                    <xsd:complexType name="checkNumber">
                        <xsd:sequence>
                            <xsd:element name="arg0" type="xsd:int"/>
                        </xsd:sequence>
                    </xsd:complexType>
                    <xsd:element name="checkNumber" type="tns:checkNumber"/>
                    <xsd:complexType name="checkNumberResponse">
                        <xsd:sequence>
                            <xsd:element name="return" type="xsd:boolean"/>
                        </xsd:sequence>
                    </xsd:complexType>
                    <xsd:element name="checkNumberResponse"
type="tns:checkNumberResponse"/>
                </schema>
            </wsdl:types>
        <wsdl:message name="addInput">
            <wsdl:part name="parameters" element="tns:add"/>
        </wsdl:message>
        <wsdl:message name="addOutput">
            <wsdl:part name="parameters" element="tns:addResponse"/>
        </wsdl:message>
        <wsdl:message name="squareInput">
            <wsdl:part name="parameters" element="tns:square"/>
        </wsdl:message>
        <wsdl:message name="squareOutput">
            <wsdl:part name="parameters" element="tns:squareResponse"/>
        </wsdl:message>
        <wsdl:message name="checkNumberInput">
            <wsdl:part name="parameters" element="tns:checkNumber"/>
        </wsdl:message>
        <wsdl:message name="checkNumberOutput">
            <wsdl:part name="parameters" element="tns:checkNumberResponse"/>
        </wsdl:message>
        <wsdl:message name="CalculatorWrapperException">
            <wsdl:part name="CalculatorWrapperException"
element="ns0:CalculatorWrapperException"/>
        </wsdl:message>
        <wsdl:message name="CalculatorException">
            <wsdl:part name="CalculatorException" element="ns0:CalculatorException"/>
        </wsdl:message>
        <wsdl:portType name="Calculator">
```

```
        <wsdl:operation name="add">
            <wsdl:input xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
message="tns:addInput" ns1:Action=""/>
            <wsdl:output xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
message="tns:addOutput" ns1:Action=""/>
        </wsdl:operation>
        <wsdl:operation name="square">
            <wsdl:input xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
message="tns:squareInput" ns1:Action=""/>
            <wsdl:output xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
message="tns:squareOutput" ns1:Action=""/>
        </wsdl:operation>
        <wsdl:operation name="checkNumber">
            <wsdl:input xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
message="tns:checkNumberInput" ns1:Action=""/>
            <wsdl:output xmlns:ns1="http://www.w3.org/2006/05/addressing/wsdl"
message="tns:checkNumberOutput" ns1:Action=""/>
            <wsdl:fault name="CalculatorWrapperException"
message="tns:CalculatorWrapperException"/>
            <wsdl:fault name="CalculatorException"
message="tns:CalculatorException"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="CalculatorSoapHttp" type="tns:Calculator">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/
soap/http"/>
        <wsdl:operation name="add">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="square">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="checkNumber">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="CalculatorWrapperException">
                <soap:fault name="CalculatorWrapperException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
            <wsdl:fault name="CalculatorException">
                <soap:fault name="CalculatorException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
```

```
            </wsdl:binding>
        <wsdl:service name="CalculatorService">
            <wsdl:port name="CalculatorPort" binding="tns:CalculatorSoapHttp">
                <soap:address location="http://localhost:8888/CalWSBA/
CalculatorPort"/>
            </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

The following example uses the WSDL file shown in the previous example. First, the tool parses the WSDL for all port types and bindings (supported protocols such as HTTP get/post or SOAP) to list all the operations. If there are multiple port types in the WSDL, the tool first prompts you to select a port type.

To start the command-line tool:

1. Go to the $AGENT_HOME/bin directory.

2. Run the following command:

   ```
   $ emctl wscli -metadata -wsdl=/tmp/CalculatorWS.wsdl
   ```

Once invoked, the command-line tool automatically prompts you for the requisite information, as shown in the following example. If you need to quit a command-line tool session, you can press Ctrl+C at any point to exit. Session information will not be saved.

**Example: Sample Web Services Command-Line Tool Session**

```
Oracle Enterprise Manager 12c Release 1 Cloud Control 12.1.0.1.0
Copyright (c) 1996, 2011 Oracle Corporation.  All rights reserved.

OracleHome : /oracle/oms/agent
EMDROOT    : /oracle/oms/agent

Generate Metric Metadata for Web Service Monitoring

Reading WSDL Document at /tmp/CalculatorWS.wsdl...done.

==> Enter the metadata file name [/tmp/target/metadata/CalculatorService.xml] :

* Selected Service: CalculatorService

* Selected Port: CalculatorPort

All operations for the selected Port "CalculatorPort":
[1]    squareResponse square(int arg0)
[2]    checkNumberResponse checkNumber(int arg0)
[3]    addResponse add(int arg0, int arg1)

==> Enter the index [1-3] of operation to select: 1
* Selected Operation:
    squareResponse square(int arg0)

Define new metric group:
==> Enter the name for this metric group [square]:

Return value(s) for the selected operation:
[1]    //ns0:squareResponse/arg0 <int>
```

```
==> Enter the index [1-1] of metric to display: 1
==> Enter the name for this metric [arg0]: SquareResult
==> Enter the label for this metric [SquareResult]:
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for this item <y/n>? [n] :


Setup operation Argument: square.arg0 <type:int>
==>   Enter value [%square.arg00001%] :

==> Do you want to use jps-config-jse.xml <y/n>? [n] :

==> Do you want to add User/Password Credential <y/n>? [n] : y
==> Enter the name for User/Password credential set [UserCredentialSet01] :

==> Do you want to add SSL TrustStore Credential <y/n>? [n] :

==> Do you want to add SSL KeyStore Credential <y/n>? [n] :

==> Do you want to add KeyStore Credential <y/n>? [n] :

==> Do you want to add Encryption Key Credential <y/n>? [n] :

==> Do you want to add Signature Key Credential <y/n>? [n] :

==> Is this metric group for periodic collection <y/n>? [y] :
The following units are for collection frequency:
[1]   Min
[2]   Hr
[3]   Day

==> Enter the index [1-3] of unit for this collection: 1
==> Enter the frequency of collection in Min: 30

==> Do you want to add another metric group <y/n>? [n] :

Files Generated:
- Target Metadata file: /tmp/target/metadata/CalculatorService.xml
- Target Collection file: /tmp/target/metadata/CalculatorServiceCollection.xml
```

The command-line tool generates the metadata required to monitor the
CalculatorService target type as shown in the following example.

**Example: CalculatorService Target Metadata File**

```
<!DOCTYPE TargetMetadata SYSTEM "../dtds/TargetMetadata.dtd">
<TargetMetadata META_VER="1.0" TYPE="CalculatorService">
  <Display>
    <Label NLSID="NLSID_CALCULATOR_SERVICE">CalculatorService</Label>
    <ShortName NLSID="NLSID_CALCULATOR_SERVICE">CalculatorService</ShortName>
    <Description NLSID="NLSID_CALCULATOR_SERVICE">CalculatorService</Description>
  </Display>
  <Metric NAME="square" TYPE="TABLE">
    <Display>
      <Label NLSID="NLSID_SQUARE">square</Label>
    </Display>
    <TableDescriptor>
      <ColumnDescriptor IS_KEY="FALSE" NAME="SquareResult" TYPE="STRING">
        <Display>
          <Label NLSID="COL_SQUARE_RESULT">SquareResult</Label>
```

```
            </Display>
        </ColumnDescriptor>
      </TableDescriptor>
      <QueryDescriptor FETCHLET_ID="WSF">
        <Property NAME="ProxyHost" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyHost</
Property>
        <Property NAME="ProxyPort" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyPort</
Property>
        <Property NAME="SecurityPolicy" SCOPE="INSTANCE"
OPTIONAL="FALSE">square.SecurityPolicy</Property>
        <Property NAME="ServiceEndpoint" SCOPE="INSTANCE"
OPTIONAL="FALSE">square.ServiceEndpoint</Property>
        <Property NAME="ServiceName" SCOPE="GLOBAL"
OPTIONAL="FALSE">ns0:CalculatorService</Property>
        <Property NAME="PortName" SCOPE="GLOBAL"
OPTIONAL="FALSE">ns0:CalculatorPort</Property>
        <Property NAME="OperationName" SCOPE="GLOBAL" OPTIONAL="FALSE">square</
Property>        <Property NAME="MessageType" SCOPE="GLOBAL"
OPTIONAL="FALSE">SOAP</Property>
        <Property NAME="SOAPBindingStyle" SCOPE="GLOBAL"
OPTIONAL="FALSE">DOCUMENT</Property>
        <Property NAME="SOAPBindingUse" SCOPE="GLOBAL" OPTIONAL="FALSE">LITERAL</
Property>
        <Property NAME="ParameterStyle" SCOPE="GLOBAL" OPTIONAL="FALSE">WRAPPED</
Property>
        <Property NAME="SOAPVersion" SCOPE="GLOBAL" OPTIONAL="FALSE">SOAP_1_1</
Property>
        <Property NAME="Namespace" SCOPE="GLOBAL" OPTIONAL="FALSE"><!
[CDATA[[ns0="http://tests.jaxws.oracle.com/"]]]></Property>
        <Property NAME="RowType" SCOPE="GLOBAL" OPTIONAL="FALSE">//
ns0:squareResponse/arg0</Property>
        <Property NAME="ColType" SCOPE="GLOBAL"
OPTIONAL="FALSE">SquareResult:STRING</Property>
        <Property NAME="Payload" SCOPE="GLOBAL" OPTIONAL="FALSE"><!
[CDATA[<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
            <soap:Body xmlns:ns1="http://tests.jaxws.oracle.com/">
              <ns1:square>
                <arg0>%square.arg00001%</arg0>
              </ns1:square>
            </soap:Body>
          </soap:Envelope>]]></Property>
        <Property NAME="UserCredential" SCOPE="GLOBAL"
OPTIONAL="FALSE">UserCredentialSet01</Property>
        <CredentialRef NAME="UserCredentialSet01">UserCredentialSet01</
CredentialRef>
      </QueryDescriptor>
    </Metric>
    <CredentialInfo>
      <CredentialType NAME="CSFKeyCredential">
        <Display>
          <Label NLSID="CRED_TYPE">CSF-Key Credential Type</Label>
        </Display>
        <CredentialTypeColumn NAME="CSFKey">
          <Display>
            <Label NLSID="CRED_C_S_F_KEY">Alias CSF Key</Label>
          </Display>
        </CredentialTypeColumn>
      </CredentialType>
      <CredentialType NAME="AliasCredential">
        <Display>
          <Label NLSID="CRED_TYPE">Alias Credential Type</Label>
```

```
                </Display>
            <CredentialTypeColumn NAME="Alias">
              <Display>
                <Label NLSID="CRED_ALIAS">Alias (i.e. username, encryption key,
signature key, etc)</Label>
              </Display>
            </CredentialTypeColumn>
            <CredentialTypeColumn NAME="Password">
              <Display>
                <Label NLSID="CRED_PASSWORD">Password for the alias</Label>
              </Display>
            </CredentialTypeColumn>
          </CredentialType>
          <CredentialSet NAME="UserCredentialSet01" USAGE="MONITORING">
            <AllowedCredType TYPE="CSFKeyCredential"/>
            <AllowedCredType TYPE="AliasCredential"/>
          </CredentialSet>
        </CredentialInfo>
        <InstanceProperties>
          <InstanceProperty NAME="ProxyHost" CREDENTIAL="FALSE" OPTIONAL="TRUE">
            <Display>
              <Label NLSID="PROP_PROXY_HOST">Proxy Server Name</Label>
            </Display>
          </InstanceProperty>
          <InstanceProperty NAME="ProxyPort" CREDENTIAL="FALSE" OPTIONAL="TRUE">
            <Display>
              <Label NLSID="PROP_PROXY_PORT">Proxy Server Port</Label>
            </Display>
          </InstanceProperty>
          <InstanceProperty NAME="square.SecurityPolicy" CREDENTIAL="FALSE"
OPTIONAL="FALSE">
            <Display>
              <Label NLSID="PROP_SQUARE_SECURITY_POLICY">[square] Authentication/Web
Service Policy</Label>
            </Display>
          </InstanceProperty>
          <InstanceProperty NAME="square.ServiceEndpoint" CREDENTIAL="FALSE"
OPTIONAL="FALSE">
            <Display>
              <Label NLSID="PROP_SQUARE_SERVICE_ENDPOINT">[square] Web Service
Endpoint URL</Label>
            </Display>
          </InstanceProperty>
          <InstanceProperty NAME="square.arg00001" CREDENTIAL="FALSE" OPTIONAL="FALSE">
            <Display>
              <Label NLSID="PROP_SQUARE_ARG00001">[square] square.arg0</Label>
          </Display>    </InstanceProperty>  </InstanceProperties></TargetMetadata>
```

The command-line tool also generates the requisite collection file as shown in the
following example.

**Example: CalculatorService Default Collection File**

```
<!DOCTYPE TargetCollection SYSTEM "../dtds/TargetCollection.dtd">
<TargetCollection TYPE="CalculatorService">
  <CollectionItem NAME="square">
    <Schedule>
      <IntervalSchedule TIME_UNIT="Min" INTERVAL="30"/>
    </Schedule>
  </CollectionItem>
</TargetCollection>
```

After the tool generates the target metadata and collection files, you can create the Oracle Plug-in archive. For more information, see Creating the Plug-in Archive.

# Monitoring Using WS-Management in Enterprise Manager

Beginning with Enterprise Manager 12*c*, WS-Management (WS-MAN)-compliant resources can be monitored using the fetchlet WSManagementFetchlet.

The fetchlet communicates with the WS-MAN resources using WS-Transfer protocol, which defines a number of management operations that the managed resources should support. However, in the current release, the fetchlet only supports the operation WS-Transfer GET.

> **Note:**
>
> For more information about the monitor WS-Management standard, see the DMTF Web Services Management website:
>
> http://www.dmtf.org/standards/wsman

**Prerequisites**

- Management Agent version 12.1.0.0.0 or greater installed on that host.
- Oracle Management Server (OMS) version 12.1.0.0.0 or greater with which the Management Agent communicates.

## Creating Metadata and Default Collection Files

Enterprise Manager provides an easy-to-use WS-Management CLI command-line tool that simplifies creating new Management Plug-ins by automatically generating the requisite target metadata and default collection files. Information retrieval is achieved via the WSManagementFetchlet that is integrated with the Management Agent.

Resources, which support WS-Management interface, should describe their model-specific elements using XML Schema Definition (XSD) representation and expose the XSD as a public accessible link just like WSDL for Web services.

The command-line tool works by parsing a specified XSD file for the managed WS-MAN resource and then prompts you to select the interested resource properties to construct a monitoring metric.

## WS-Management CLI Command-line Tool Syntax

The WS-Management CLI command-line tool syntax is as follows:

```
Usage: emctl wsmancli [-metadata | -help] [-options]
```

The command accepts the following options:

- *-schema=file | URL*: Resource XSD file or URL [mandatory]
- *-username=user ID* : User name if the schema is protected

The command-line tool requires a XSD file name or URL to locate the resource schema. For example, for a Traffic Light WS-Management service, a XSD URL would be as follows:

```
http://localhost:8888/TrafficLight?xsd
```

The command tool script requires access to the Enterprise Manager home directory (EM_HOME) to run. The tool defaults to ORACLE_HOME (ensure this environment variable is set properly before using this tool).

## Command-line Tool Security

The command-line tool generates metadata required by Enterprise Manager for target monitoring purposes via the resource XSD. When you run this tool, you only need read permission on the XSD file or URL and permission to save generated files to the appropriate directory.

## Generating Target Metadata and Collection Files

The following example shows a sample XSD file passed to the command-line tool to generate the target metadata and collection files.

**Example: Sample XSD File TrafficLight.xsd**

```
<?xml version="1.0" encoding="UTF-8"?><xs:schema targetNamespace="http://
schemas.wiseman.dev.java.net/traffic/1/light.xsd" elementFormDefault="qualified"
blockDefault="#all" xmlns:tl="http://schemas.wiseman.dev.java.net/traffic/1/
light.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="TrafficLightType">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="color" type="xs:string"/>
            <xs:element name="x" type="xs:int"/>
            <xs:element name="y" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="trafficlight" type="tl:TrafficLightType"/>
</xs:schema>
```

To start the command-line tool:

1. Go to the $AGENT_HOME/bin directory.

2. Execute the following command:

   ```
   $ emctl wsmancli -metadata -schema= http://localhost:8080/Traffic?xsd
   ```

Once invoked, the command-line tool automatically prompts you for the requisite information, as shown in the following example. If you need to quit a command-line tool session, you can press Control+C at any point to exit. Session information will not be saved.

**Example: Sample WS-Management CLI Command-Line Tool Session**

```
Oracle Enterprise Manager 12c Release 1 Cloud Control 12.1.0.0.0
Copyright (c) 1996, 2011 Oracle Corporation.  All rights reserved.

OracleHome : /oracle/oms/agent
EMDROOT    : /oracle/oms/agent
```

Generate Metric Metadata for WS-Management Resource Monitoring

Reading Resource XSD Document at http://localhost:8080/Traffic?xsd...done.

==> Enter the name for this target type: TrafficLight

==> Enter the metadata file name [/tmp/target/metadata/TrafficLight.xml] :

Define new metric group name:
==> Enter the name for this metric group: trafficLight

WS-Addressing namespaces:
[1]    http://www.w3.org/2005/08/addressing
[2]    http://schemas.xmlsoap.org/ws/2004/08/addressing

==> Enter the index [1-2] to select: 1


SOAP Envelope namespaces:
[1]    http://www.w3.org/2003/05/soap-envelope
[2]    http://schemas.xmlsoap.org/soap/envelope/

==> Enter the index [1-2] to select: 1

Resource properties:
[1]    trafficlight:color
[2]    trafficlight:name
[3]    trafficlight:x
[4]    trafficlight:y

==> Enter the index [1-4] of property to display: 2
==> Enter the name for this metric [name]:
==> Enter the label for this metric [name]:
==> Is this a key metric <y/n>? [n] : y
==> Do you want to add another metric <y/n>? [n] : y

Resource properties:
[1]    trafficlight:color
[2]    trafficlight:x
[3]    trafficlight:y

==> Enter the index [1-3] of property to display: 1
==> Enter the name for this metric [color]:
==> Enter the label for this metric [color]:
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for this item <y/n>? [n] :
==> Do you want to add another metric <y/n>? [n] : y

Resource properties:
[1]    trafficlight:x
[2]    trafficlight:y

==> Enter the index [1-2] of property to display: 1
==> Enter the name for this metric [x]:
==> Enter the label for this metric [x]:
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for this item <y/n>? [n] :
==> Do you want to add another metric <y/n>? [n] : y

Resource properties:
[1]    trafficlight:y

```
==> Enter the index [1-1] of property to display: 1
==> Enter the name for this metric [y]:
==> Enter the label for this metric [y]:
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for this item <y/n>? [n] :


==> Enter comma-separated list of Selector elements: name

==> Do you want to add User/Password Credential <y/n>? [n] : y
==> Enter the name for User/Password credential set [UserCredentialSet01] :

==> Is this metric group for periodic collection <y/n>? [y] :
The following units are for collection frequency:
[1]   Min
[2]   Hr
[3]   Day

==> Enter the index [1-3] of unit for this collection: 1
==> Enter the frequency of collection in Min: 30

==> Do you want to add another metric group <y/n>? [n] :

Files Generated:
- Target Metadata file: /tmp/target/metadata/TrafficLight.xml
- Target Collection file: /tmp/target/metadata/TrafficLightCollection.xml
```

The command-line tool generates the metadata required to monitor the target type
TrafficLight as shown in the following example.

**Example: TrafficLight Target Metadata File**

```
<!DOCTYPE TargetMetadata SYSTEM "../dtds/TargetMetadata.dtd">
<TargetMetadata META_VER="1.0" TYPE="TrafficLight">
  <Display>
    <Label NLSID="NLSID_TRAFFIC_LIGHT">TrafficLight</Label>
    <ShortName NLSID="NLSID_TRAFFIC_LIGHT">TrafficLight</ShortName>
    <Description NLSID="NLSID_TRAFFIC_LIGHT">TrafficLight</Description>
  </Display>
  <Metric NAME="trafficLight" TYPE="TABLE">
    <Display>
      <Label NLSID="NLSID_TRAFFIC_LIGHT">trafficLight</Label>
    </Display>
    <TableDescriptor>
      <ColumnDescriptor IS_KEY="TRUE" NAME="name" TYPE="STRING">
        <Display>
          <Label NLSID="COL_NAME">name</Label>
        </Display>
      </ColumnDescriptor>
      <ColumnDescriptor IS_KEY="FALSE" NAME="color" TYPE="STRING">
        <Display>
          <Label NLSID="COL_COLOR">color</Label>
        </Display>
      </ColumnDescriptor>
      <ColumnDescriptor IS_KEY="FALSE" NAME="x" TYPE="STRING">
        <Display>
          <Label NLSID="COL_X">x</Label>
        </Display>
      </ColumnDescriptor>
      <ColumnDescriptor IS_KEY="FALSE" NAME="y" TYPE="STRING">
```

```
          <Display>
            <Label NLSID="COL_Y">y</Label>
          </Display>
        </ColumnDescriptor>
      </TableDescriptor>
      <QueryDescriptor FETCHLET_ID="WSManagementFetchlet">
        <Property NAME="ProxyHost" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyHost</
Property>
        <Property NAME="ProxyPort" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyPort</
Property>
        <Property NAME="SecurityPolicy" SCOPE="INSTANCE"
OPTIONAL="TRUE">trafficLight.SecurityPolicy</Property>
        <Property NAME="ResourceURL" SCOPE="INSTANCE"
OPTIONAL="FALSE">trafficLight.ResourceURL</Property>
        <Property NAME="To" SCOPE="INSTANCE" OPTIONAL="FALSE">trafficLight.To</
Property>
        <Property NAME="OptionSet" SCOPE="INSTANCE"
OPTIONAL="TRUE">trafficLight.OptionSet</Property>
        <Property NAME="Locale" SCOPE="INSTANCE"
OPTIONAL="TRUE">trafficLight.Locale</Property>
        <Property NAME="MaxEnvelopeSize" SCOPE="INSTANCE"
OPTIONAL="TRUE">trafficLight.MaxEnvelopeSize</Property>
        <Property NAME="OperationTimeout" SCOPE="INSTANCE"
OPTIONAL="TRUE">trafficLight.OperationTimeout</Property>
        <Property NAME="Namespace"
SCOPE="GLOBAL" OPTIONAL="FALSE"><![CDATA[[ns1="http://
schemas.wiseman.dev.java.net/traffic/1/light.xsd"][ns0="http://www.w3.org/2001/
XMLSchema"][wsa="http://www.w3.org/2005/08/addressing"][env="http://www.w3.org/
2003/05/soap-envelope"]]]></Property>
        <Property NAME="RowType" SCOPE="GLOBAL" OPTIONAL="FALSE">
//ns1:trafficlight/ns1:name,//ns1:trafficlight/ns1:color,//ns1:trafficlight/
ns1:x,//ns1:trafficlight/ns1:y</Property>
        <Property NAME="ColType" SCOPE="GLOBAL"
OPTIONAL="FALSE">name:STRING,color:STRING,x:STRING,y:STRING</Property>
        <Property NAME="ReplyTo" SCOPE="GLOBAL" OPTIONAL="FALSE">http://www.w3.org/
2005/08/addressing/role/anonymous</Property>
        <Property NAME="Action" SCOPE="GLOBAL" OPTIONAL="FALSE">http://
schemas.xmlsoap.org/ws/2004/09/transfer/Get</Property>
        <Property NAME="TransferOperation" SCOPE="GLOBAL" OPTIONAL="FALSE">GET</
Property>
        <Property NAME="SelectorSet" SCOPE="GLOBAL"
OPTIONAL="FALSE">[name,%trafficLight.name%]</Property>
        <Property NAME="UserCredential" SCOPE="GLOBAL"
OPTIONAL="FALSE">UserCredentialSet01</Property>
        <CredentialRef NAME="UserCredentialSet01">UserCredentialSet01</
CredentialRef>
      </QueryDescriptor>
    </Metric>
    <CredentialInfo>
      <CredentialType NAME="CSFKeyCredential">
        <Display>
          <Label NLSID="CRED_TYPE">CSF-Key Credential Type</Label>
        </Display>
        <CredentialTypeColumn NAME="CSFKey">
          <Display>
            <Label NLSID="CRED_C_S_F_KEY">Alias CSF Key</Label>
          </Display>
        </CredentialTypeColumn>
      </CredentialType>
      <CredentialType NAME="AliasCredential">
        <Display>
```

```
        <Label NLSID="CRED_TYPE">Alias Credential Type</Label>
      </Display>
      <CredentialTypeColumn NAME="Alias">
        <Display>
          <Label NLSID="CRED_ALIAS">Alias (i.e. username, encryption
key, signature key, etc)</Label>
        </Display>
      </CredentialTypeColumn>
      <CredentialTypeColumn NAME="Password">
        <Display>
          <Label NLSID="CRED_PASSWORD">Password for the alias</Label>
        </Display>
      </CredentialTypeColumn>
    </CredentialType>
    <CredentialSet NAME="UserCredentialSet01" USAGE="MONITORING">
      <AllowedCredType TYPE="CSFKeyCredential"/>
      <AllowedCredType TYPE="AliasCredential"/>
    </CredentialSet>
  </CredentialInfo>
  <InstanceProperties>
    <InstanceProperty NAME="ProxyHost" CREDENTIAL="FALSE" OPTIONAL="TRUE">
      <Display>
        <Label NLSID="PROP_PROXY_HOST">Proxy Server Name</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="ProxyPort" CREDENTIAL="FALSE" OPTIONAL="TRUE">
      <Display>
        <Label NLSID="PROP_PROXY_PORT">Proxy Server Port</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.SecurityPolicy"
      CREDENTIAL="FALSE" OPTIONAL="TRUE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_SECURITY_POLICY">[trafficLight]
Authentication/Web Service Policy</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.ResourceURL" CREDENTIAL="FALSE"
OPTIONAL="FALSE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_RESOURCE_U_R_L">[trafficLight] Resource
URL (wsman:ResourceURL)</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.To" CREDENTIAL="FALSE" OPTIONAL="FALSE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_TO">[trafficLight] Network Address
of the service (wsa:To)</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.OptionSet" CREDENTIAL="FALSE"
OPTIONAL="TRUE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_OPTION_SET">[trafficLight] Set
of wsman:Option. Format: [&lt;OptionName1&gt;, value:&lt;value1&gt;,
type:&lt;type1&gt;, mustComply:&lt;true|false&gt;][&lt;OptionName2&gt;,
value:&lt;value2&gt;, type:&lt;type&gt;, mustComply:&lt;true|false&gt;][...]</
Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.Locale" CREDENTIAL="FALSE"
```

```
OPTIONAL="TRUE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_LOCALE">[trafficLight] wsman:Locale
(RFC 3066 language code). Format: e.g. en-US</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.MaxEnvelopeSize"
      CREDENTIAL="FALSE" OPTIONAL="TRUE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_MAX_ENVELOPE_SIZE">[trafficLight]
wsman:MaxEnvelopeSize in Octets. Format: e.g. 8192</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.OperationTimeout"
      CREDENTIAL="FALSE" OPTIONAL="TRUE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_OPERATION_TIMEOUT">[trafficLight]
wsman:OperationTimeout. Format: e.g. PT30S</Label>
      </Display>
    </InstanceProperty>
    <InstanceProperty NAME="trafficLight.name" CREDENTIAL="FALSE"
OPTIONAL="FALSE">
      <Display>
        <Label NLSID="PROP_TRAFFIC_LIGHT_NAME">[trafficLight] Value for
the Selector "name"</Label>
      </Display>
    </InstanceProperty>
  </InstanceProperties>
</TargetMetadata>
```

The command-line tool also generates the requisite collection file as shown in the following example.

**Example: TrafficLight Default Collection File**

```
<!DOCTYPE TargetCollection SYSTEM "../dtds/TargetCollection.dtd">
<TargetCollection TYPE="TrafficLight">
  <CollectionItem NAME="trafficLight">
    <Schedule>
      <IntervalSchedule TIME_UNIT="Min" INTERVAL="30"/>
    </Schedule>
  </CollectionItem>
</TargetCollection>
```

After the command-line tool generates the target metadata and collection files, you can create the Metatdata Plug-in archive. See Creating the Plug-in Archive.

# Monitoring a Standalone JMX-instrumented Java Application or JVM Target

> **✎ Note:**
>
> If your Java application is not JMX-instrumented, but you want to monitor the J2SE 1.5 or higher JVM on which it is running, go directly to Configuring a Standalone Java Application or JVM Target to create target instances of type JVM. This enables you to monitor these JVMs in Enterprise Manager, preferably from an Enterprise Manager Agent installed on the same host as your JVM. However, the prerequisites and known limitations discussed below still apply.

Enterprise Manager provides an out-of-box JVM target type. This enables you to add and configure metrics from standalone J2SE1.5 JVMs that are enabled for remote management in Enterprise Manager version 10.2.0.3 or later.

If your standalone Java application exposes data through JMX MBeans as for a J2EE application deployed on an Oracle Container for J2EE, you can use the JMX command-line tool to define such an application as an Enterprise Manager target type and generate a metadata and default collection file for this target type. You can monitor your standalone application targets from an Enterprise Manager Agent, preferably installed on the same host as your JVM. Multiple JVMs running on that host can be monitored by the same Enterprise Manager Agent.

You can collect metrics from user-defined MBeans on a standalone J2SE1.5 or higher JVM and place them into Enterprise Manager using the JMX fetchlet. The fetchlet is designed for a standalone Sun J2SE1.5 or higher JVM containing user-defined MBeans that use JMX OpenTypes as arguments and return values.

**Prerequisites**

- Java virtual machine J2SE 1.5 or higher instance running on a specific host. This JVM could be running a JMX-enabled application that exposes metrics via MBeans that need to be monitored as a target in Enterprise Manager. If the application does not expose MBeans, the JVM itself could be monitored using the built-in JVM target type provided in Enterprise Manager. See Configuring a Standalone Java Application or JVM Target for more information.

- Monitoring and management from remote systems enabled. Set this system property when you start the JVM:

  ```
  com.sun.management.jmxremote.port=portNum
  ```

  For additional information about enabling the JVM for remote management, see the following document:

  ```
  http://docs.oracle.com/javase/7/docs/technotes/guides/management/
  agent.html
  ```

- Management Agent version 10.2.0.3 or later installed on that host.

- Oracle Management Server (OMS) version 10.2.0.3 or greater with which the Management Agent communicates.

**Known Limitations**

Currently, the jmxcli tool only allows you to browse and monitor MBeans (platform and application-defined) that are available on the default platform MBeanserver on the target JVM instance. The tool does not support monitoring a custom MBeanServer on the target JVM instance. The jmxcli tool primarily handles attributes as well as parameter and return values for operations that are OpenTypes, such as SimpleTypes, CompositeTypes, TabularTypes, and arrays of SimpleTypes.

# Generating Metadata and Default Collection Files

As with Web services, the command-line tool (jmxcli) simplifies creating the requisite target definition files: metadata and the default collection file for a standalone JMX-instrumented Java application. The tool is an offline configuration utility that connects you to an MBeanServer on a J2SE1.5 or higher JVM and enables you to browse available MBeans. It can also append metrics to an existing set of files during a subsequent invocation of the tool.

During a command-line tool session, you select specific MBeans and then choose the desired attributes/statistical values or operations Enterprise Manager needs to retrieve or invoke periodically on these MBeans to collect these values. The tool helps define packaging for these collected values as one or more Enterprise Manager metrics (with columns), and also enables you to specify a metric collection interval.

# JMX Command-line Tool Syntax

The JMX command-line tool syntax is as follows:

```
cd Agent Instance dir/bin
emctl jmxcli -t JVM
                [  -l JMXServiceURL
                   -h hostname
                   -p port
                   -u username
                   -c credential/password
                   -w work directory
                   -e true/false
                   [-m MBeanName | -d jmx_domain | -s mBeanPattern]
                ]
```

The `jmxcli` command accepts the following options:

- **-t JVM** Indicates that the MBeanServer is on a standalone JVM

- **-l** JMXServiceURL of the target JVM

- **-h** Hostname of the JVM. Default: "localhost" if the -l option is not specified

- **-p** RMI/RMIS port of the JVM. Default: "23791" if the -l option is not specified. From the `ORACLE_HOME/opmn/bin` directory of your Application Server 10.1.3.0 or later instance, run `opmnctl status -l` to determine the RMI port for the OC4J for which MBeans were deployed.

- **-u** Valid user name for the JVM. Default: None

- **-c** Password for the above user. Default: None. The password is only used to retrieve data and is not stored anywhere.

- **-w** Work directory where the metadata and default collection files are created. Default: Current directory. When invoking the command-line tool, you must have write permission on this directory to create subdirectories and add files. If the metadata and default collection files already exist within that directory, you have the option of appending to or overwriting the original files.

- **-e** True for enabling the SSL connection to the JVM. Default: false

You can also specify ONE of the following three parameters (`-m`, `-d` or `-s`) to retrieve a subset of MBeans available on the MBeanServer. By default, all MBeans on the MBeanServer are displayed for you to select from if none of these parameters are specified.

- **-m** MBean ObjectName of the required MBean that needs to be retrieved and examined. If this is an ObjectName pattern-matching multiple MBeans, you are shown a list of all MBeans that match the pattern, and you can select one at a time to work on.

- **-d** MBean domain of the required application whose MBeans need to be retrieved and examined. For example, you want to browse all MBeans for an application (myApp). MBeans for this application would be available in the JMX domain "myApp".

- **-s** MBean pattern matching an set of similar MBeans from which the metrics are to be defined. The `-s` parameter allows bulk retrieval of JMX Attributes/Statistics from multiple MBeans of a similar type.

  If you specify the `-s` parameter, the resulting metrics created during this `jmxcli` session appear as a table in the Enterprise Manager console with multiple rows — one row representing each MBean that matches the specified pattern, and with the MBean ObjectName as a key column. For example, if you specify `-s 'oc4j:j2eeType=Servlet,*'` the resulting metric will have multiple rows, one for each servlet that matches the ObjectName pattern. Besides the MBean ObjectName column, other columns would be the attributes or fields from the return object of the operation, selected during the `jmxcli` session.

## Generating the Files

The following steps explain how to prepare for and then use the JMX command-line tool to generate the files.

1. Bring up the standalone JVM instance with the MBeans. The following example shows an invocation of the JVM:

   ```
   JDK15/bin/java com.sun.management.jmxremote
   com.sun.management.jmxremote.port=6789
   com.sun.management.jmxremote
   com.sun.management.jmxremote.authenticate=false
   com.sun.management.jmxremote.ssl=false MyJMXEnabledApp $*
   ```

   The `jmxcli` tool connects to the port number above as a JSR-160 client.

2. Go to the `$ORACLE_HOME/bin` directory of the 10.2.0.3 or later version of the Enterprise Manager Agent.

3. Set the environment variable as follows:

   ```
   setenv USER_JARS /myAppHome/myJar1.jar;/myAppHome/myJar2.jar
   ```

This step is needed if custom classes are being returned in attributes and/or operations in any of the MBeans registered with the target MBeanServer. The Enterprise Manager Agent (fetchlet) can only effectively monitor attributes and/or operations that return JMX OpenTypes.

> **Note:**
>
> If the application-defined MBeans are returning custom classes, you need to also set up the corresponding user jar file in the `CLASSPATH` of the Enterprise Manager Agent monitoring this application. To do this, manually insert the location of this jar into the `$ORACLE_HOME/sysman/config/classpath.lst` file.

4. Run the following command:

```
./emctl jmxcli -t JVM -h localhost -p 6789 u user -c password
```

where:

- **-t** JVM indicates that the MBeanServer is running on a standard JVM
- **-h** Host name where the JVM is running
- **-p** Port number that enables the JVM for JSR-160 remote access

You can also specify an `-l JMXServiceURL` option instead of `-h host` and `-p port` options.

You can invoke `jmxcli` with a `-w work directory` option to create the metadata and default collection files in the specified work directory. If you do not specify `-w` when you start `jmxcli`, it defaults to the current directory, which is the directory where you start `jmxcli`.

Once invoked, the command-line interface automatically prompts you for the requisite information, as shown in the Sample JMXCLI Session example. For most of the prompts, you can just press enter to use defaults. If you need to quit a JMX command-line tool session, you can press Control+C at any point to exit. Session information will not be saved.

When the session concludes after you exit, the result will be a `myJ2EEApp.xml` file (or whatever target type you specified) as `metadata/myJ2EEApp.xml`, and a `default_collection/myJ2EEApp.xml` file if you specified periodic collection.

**Sample JMXCLI Invocations**

The following sample enables you to browse all MBeans on a remote MBeanServer:

```
./emctl jmxcli -t JVM -p 6789 (the host defaults to "localhost")
```

The following sample invokes the command-line interface and filters MBeans based on the MBeanPattern specified as the argument for the `-m` option:

```
./emctl jmxcli -t JVM -p 6789 -m "java.lang:*"
```

**Example: Sample JMXCLI Session**

```
oracleHome=/ade/sparmesw_emas_ml/oracle
userJars=
```

```
Connecting to server: localhost:6789
Connecting without authentication. For specifying username and password use
the -u and -c options.

Obtained 14 MBeans matching pattern java.lang:*.

Enter the target type for this metric: [myJ2EEApp] myJavaApp

Enter the target version: [1.0]

Enter the target metadata file: [./metadata/myJavaApp.xml]

Enter the default collections file: [./default_collection/myJavaApp.xml]

Enter a label for this target type: [myJavaApp]

Enter a description for this target type: [myJavaApp]
The available targets are:
0: sun.management.CompilationImpl
        (java.lang:type=Compilation)

1: sun.management.MemoryManagerImpl
(java.lang:name=CodeCacheManager,type=MemoryManager)

2: sun.management.GarbageCollectorImpl
(java.lang:name=Copy,type=GarbageCollector)

3: sun.management.MemoryPoolImpl        (java.lang:name=Eden
Space,type=MemoryPool)

4: sun.management.RuntimeImpl        (java.lang:type=Runtime)

5: sun.management.ClassLoadingImpl        (java.lang:type=ClassLoading)

6: sun.management.MemoryPoolImpl        (java.lang:name=Survivor
Space,type=MemoryPool)

7: sun.management.ThreadImpl        (java.lang:type=Threading)

8: sun.management.GarbageCollectorImpl
(java.lang:name=MarkSweepCompact,type=GarbageCollector)

9: com.sun.management.UnixOperatingSystem        (java.lang:type=OperatingSystem)

10: sun.management.MemoryImpl        (java.lang:type=Memory)

11: sun.management.MemoryPoolImpl        (java.lang:name=Code
Cache,type=MemoryPool)

12: sun.management.MemoryPoolImpl        (java.lang:name=Tenured
Gen,type=MemoryPool)

13: sun.management.MemoryPoolImpl        (java.lang:name=Perm
Gen,type=MemoryPool)

Enter the index of target/MBean you wish to monitor or press <Ctrl-C> to quit: 4

Following metric source types are available for selected target(s):
        0: JMX Attributes

Enter the index of your choice or press <Ctrl-C> to quit: 0
```

```
            Attributes are:

                0: BootClassPath         Return Value: java.lang.String
                1: BootClassPathSupported        Return Value: boolean
                2: ClassPath      Return Value: java.lang.String
                3: InputArguments        Return Value: [Ljava.lang.String;
                4: LibraryPath    Return Value: java.lang.String
                5: ManagementSpecVersion         Return Value: java.lang.String
                6: Name  Return Value: java.lang.String
                7: SpecName       Return Value: java.lang.String
                8: SpecVendor     Return Value: java.lang.String
                9: SpecVersion    Return Value: java.lang.String
                10: StartTime     Return Value: long
                11: SystemProperties      Return Value:
javax.management.openmbean.TabularData
                12: Uptime        Return Value: long
                13: VmName        Return Value: java.lang.String
                14: VmVendor      Return Value: java.lang.String
                15: VmVersion     Return Value: java.lang.String
Select one or more items as comma-separated indices: 6,7,8


Number of possible columns in the resultant metric are 3.


Enter the name for this metric column at index=0 : [Name]
Is this column a KEY Column <y/n>? [n] y
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [Name]
Enter the NLSID for column: [Name]
Enter the UNIT for column "Name": [millisec, kb etc.. ]


Enter the name for this metric column at index=1 : [SpecName]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [SpecName]
Enter the NLSID for column: [SpecName]
Enter the UNIT for column "SpecName": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n]


Enter the name for this metric column at index=2 : [SpecVendor]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [SpecVendor]
Enter the NLSID for column: [SpecVendor]
Enter the UNIT for column "SpecVendor": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n]


Enter the name of this metric: RuntimeMetric
Enter the label for this metric: [RuntimeMetric]


Do you want periodic collection for this metric <y/n>? [n] y
Enter the collection interval in seconds: 300
Periodic collection interval is: 300 seconds.


Do you want to create another metric <y/n>? [n]
Written the metadata xml file: ./metadata/myJavaApp.xml.
Creating new file: ./default_collection/myJavaApp.xml.
Updated the default collection file for myJavaApp at location
./default_collection/myJavaApp.xml.
Exiting...
```

## Using the Metadata and Default Collection Files

Look at the `currentDir/metadata` and `currentDir/default_collection` directories to see the `myTarget.xml` files (for the target type you specified earlier).

You can use these files as follows:

- Convert the files to an Oracle Plug-in Archive (OPAR) and push them from OMS to the Agent and target instances created from OMS. See Validating, Packaging, and Deploying the Plug-in and Configuring a Standalone Java Application or JVM Target.

- Edit the files, extract the metric definitions and `QueryDescriptors`, move them to other metadata and default collection files, and post-process them by creating `ExecutionDescriptors` as needed.

If you want the status information of your targets to appear correctly in the Enterprise Manager console, you need to define a Response metric. See Displaying Target Status Information for more information.

# Monitoring JMX Applications Deployed on Oracle WebLogic Application Servers

The JMX fetchlet, supplied with 11.1 Management Agents, enables you to monitor key metrics in your JMX-instrumented applications deployed on Oracle WebLogic Application Server 9.x or later.Monitoring JMX-instrumented applications with Enterprise Manager entails defining a new target type that Enterprise Manager can monitor via Management Plug-ins. As with the Web services wscli command-line tool (and as was possible for Oracle Application Servers (OC4J), Enterprise Manager provides an jmxcli command-line tool to automate the generation of the target metadata and collection files for custom JMX instrumented applications on weblogic servers..

**Prerequisites**

- Oracle WebLogic Server 9.x or higher instance running on a specific host with a JMX-enabled application deployed on it that needs to be monitored as a target in Enterprise Manager.

- Management Agent version 11.1 or greater installed (preferably) on that host.

- Oracle Management Server (OMS) version 10.2.0.4 or greater with which the Management Agent communicates.

- The `jmxcli` tool primarily handles attributes and parameter and return values for operations that are OpenTypes. Examples: SimpleTypes, CompositeTypes, TabularTypes, and arrays of SimpleTypes.

## Creating Metadata and Default Collection Files using jmxcli

As with Web services, the JMX command-line tool (jmxcli) simplifies creating the requisite target definition files: metadata and the default collection file. The tool is an offline configuration utility that connects you to an MBeanServer and enables you to browse available MBeans. It can also append metrics to an existing file during a subsequent invocation of the tool.During a command-line tool session, you select

specific MBeans and then choose the desired JMX attributes/statistical values the Enterprise Manager needs to retrieve or JMX operations that need to be invoked periodically on these MBeans to collect these values. The tool helps define packaging for these collected values as one or more Enterprise Manager metrics (with columns), and also enables you to specify a metric collection interval and thresholds for the columns.

## JMX Command-line Tool Syntax

The JMX command-line tool syntax is as follows for a JMX-enabled target on an Oracle WebLogic Application Server:

```
./emctl jmxcli -t WebLogic [help|options]
  where options are:           [  -l JMX ServiceURL
              -u username
              -c credential/password
              -w work directory
              [-m MBeanName | -d jmx_domain | -s mBeanPattern]
                ]
```

The `jmxcli` command accepts the following options:

- l - JMXServiceURL to the WebLogic managed server hosting the custom MBeans in the form

  `service:jmx:t3://host:t3port/jndi/weblogic.management.mbeanservers.runtime`

- u - Valid user name for the WebLogic domain. Default: "weblogic"

- c - Password associated with the user specified by the -u option. Default: None. If you do not specify a password, you are prompted for the password.

- w - Directory where the metadata and default collection files created by the JMX command-line tool are placed. Default: Current directory.

  When invoking the command-line tool, you must have write permission on this directory to create subdirectories and add files. If the metadata and default collection files already exist within that directory, you have the option of appending to or overwriting the original files.

You can also specify ONE of the following three parameters (-m, -d or -s) to retrieve a subset of MBeans available on the MBeanServer. By default, all MBeans on the MBeanServer are displayed for you to select from if none of these parameters are specified.

- m - MBean ObjectName of the required MBean that needs to be retrieved and examined. If this is an ObjectName pattern-matching multiple MBeans, you are shown a list of all MBeans that match the pattern, and you can select one at a time to work on.

- d - MBean domain of the required application whose MBeans need to be retrieved and examined. For example, you want to browse all MBeans for an application (myApp). MBeans for this application would be available in the JMX domain "myApp".

- s - MBean pattern-matching an existing set of MBeans from which the metrics are to be defined. The -s parameter allows retrieval of JMX Attributes/Statistics from multiple MBeans of a similar type into one Metric.

If you specify the -s parameter, the resulting metrics created during this `jmxcli` session appear as a table in the Enterprise Manager console with multiple rows - one row representing each MBean that matches the specified pattern (with the MBean ObjectName as a key column if no other key columns are defined). For example, if you specify -s 'com.bea:Type=ServletRuntime,*' the resulting metric will have multiple rows, one for each servlet that matches the ObjectName pattern. Besides the MBean ObjectName key column, other columns would be the attributes or fields from the return object of the operation, selected during the `jmxcli` session.

## Generating the Files

To start the JMX command-line tool:

1. Go to the $AGENT_HOME/bin directory.

2. Run the following command:

   ```
   ./emctl jmxcli -t WebLogic [OPTIONS]
   ```

Once invoked, the command-line interface automatically prompts you for the requisite information, as shown in the following example. If you need to quit a JMX command-line tool session, you can press Control+C at any point to exit. Session information will not be saved.

The following example illustrates a sample `jmxcli` session:

**Example: jmxcli Session**

```
$ ./emctl jmxcli -t WebLogic -l "service:jmx:t3://host:22048/jndi/
weblogic.management.mbeanservers.runtime" -u weblogic -c password -s
"*:type=soainfra_bpel_requests,*"
```

*NOTE 1: The -s option above will result in a metric with as many rows as the number of MBeans which match the ObjectName pattern specified, every time the metric is collected by the agent. If you need to always collect from a specific Mbean then use the -m <ObjectName> option instead of the -s <Mbean pattern> used in above example.*

*NOTE 2: If you need to use t3s to connect to the weblogic server then the following env variable needs to be set before invoking the* `jmxcli`

```
setenv USER_JAVA_PROPS=-Dweblogic.security.TrustKeyStore=CustomTrust
 -Dweblogic.security.CustomTrustKeyStoreFileName=$ORACLE_HOME/sysman/config/
montrust/AgentTrust.jks
 -Dweblogic.security.SSL.enforceConstraints=off
 -Dweblogic.security.SSL.ignoreHostnameVerification=true
 -Djavax.net.ssl.trustStore=$ORACLE_HOME/sysman/config/montrust/AgentTrust.jks
(or set USER_JAVA_PROP= … equivalent on win32)
setenv USER_JARS  <; separated list of jars needed in classpath if custom
authentication modules are involved in SSL connection>
```

*A semi-colon is used as a delimiter for the list of jar files.*

*Example:* `setenv USER_JARS "jar1;jar2;jar3"`

*In some cases, if MBeans return custom WebLogic objects in their MBeanInfo, the weblogic.jar may need to be set to the above env variable before invoking the* `jmxcli`.

*Example:* `setenv USER_JARS $BEA_HOME/server/lib/weblogic.jar`

```
oracleHome=/ade/sparmesw_egcli/oracle/work/middleware/oms
userJars=
Connecting to server:
 service:jmx:t3://host1:22048/jndi/weblogic.management.mbeanservers.runtime
Connecting as user: weblogic
Obtained 3 MBeans matching pattern *:type=soainfra_bpel_requests,*.

Enter the target type for this metric: [myJ2EEApp] myCustomWLApp
Enter the target version: [1.0]
Enter the target metadata file: [./metadata/myCustomWLApp.xml]
Enter the default collections file: [./default_collection/myCustomWLApp.xml]
Enter a label for this target type: [myCustomWLApp]
Enter a description for this target type: [myCustomWLApp]
The available targets are:
0: DMS metric mbean
        (oracle.dms:name=/soainfra/engines/bpel/requests/
engine,type=soainfra_bpel_requests)
1: DMS metric mbean
        (oracle.dms:name=/soainfra/engines/bpel/requests/
system,type=soainfra_bpel_requests)
2: DMS metric mbean
        (oracle.dms:name=/soainfra/engines/bpel/requests/
invoke,type=soainfra_bpel_requests)
Following metric source types are available for selected target(s):
        0: JMX Attributes
Enter the index of your choice or press <Ctrl-C> to quit: 0
Attributes are:
        0: active_count  Return Value: java.lang.Integer
        1: active_maxValue       Return Value: java.lang.Integer
        2: active_minValue       Return Value: java.lang.Integer
        3: active_value  Return Value: java.lang.Integer
        4: Name  Return Value: java.lang.String
        5: Parent        Return Value: java.lang.String
        6: scheduled_count       Return Value: java.lang.Integer
        7: scheduled_maxValue    Return Value: java.lang.Integer
        8: scheduled_minValue    Return Value: java.lang.Integer
        9: scheduled_value       Return Value: java.lang.Integer
        10: threadCount_count    Return Value: java.lang.Integer
        11: threadCount_maxValue        Return Value: java.lang.Integer
        12: threadCount_minValue        Return Value: java.lang.Integer
        13: threadCount_value    Return Value: java.lang.Integer

Select one or more items as comma separated indices: 4,0,1,2
Number of possible columns in the resultant metric are 4.

Enter the name for this metric column at index=0 : [Name]
Is this column a KEY Column <y/n>? [n] y
```

*Specifying "y" signifies that the value of this column is unique in case multiple rows are returned.*

```
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [Name]
Enter the NLSID for column: [Name]
Enter the UNIT for column "Name": [millisec, kb etc.. ]

Enter the name for this metric column at index=1 : [active_count]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [active_count]
Enter the NLSID for column: [active_count]
```

```
Enter the UNIT for column "active_count": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n] y
Creating threshold!!
Following operators are available for creating thresholds:
 0: GT
 1: EQ
 2: LT
 3: LE
 4: GE
 5: CONTAINS
 6: NE
 7: MATCH

Enter the index of your choice or press <Ctrl-C> to quit: 0
Enter the CRITICAL threshold: [NotDefined] 50
Enter the WARNING threshold: [NotDefined] 45
Enter the number of occurrences that trigger threshold: [6] 3
Enter the message to be used when threshold is triggered: [active_count
is %value% and has crossed warning (%warning_threshold%) or critical
(%critical_threshold%) threshold.]
Enter NLSID for the message used when threshold is triggered:
[active_count_cond] Enter the name for this metric column at index=2 :
[active_maxValue]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [active_maxValue]
Enter the NLSID for column: [active_maxValue]
Enter the UNIT for column "active_maxValue": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n]

Enter the name for this metric column at index=3 : [active_minValue]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [active_minValue]
Enter the NLSID for column: [active_minValue]
Enter the UNIT for column "active_minValue": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n]

Enter the name of this metric: bpel_requests
Enter the label for this metric: [bpel_requests]

Do you want periodic collection for this metric <y/n>? [n] y
Enter the collection interval in seconds: 300
Periodic collection interval is: 300 seconds.

Do you want to create another metric <y/n>? [n]
Written the metadata xml file: ./metadata/myCustomWLApp.xml.
Creating new file: ./default_collection/myCustomWLApp.xml.
Updated the default collection file for myCustomWLApp at location ./
default_collection/myCustomWLApp.xml.
Exiting...
```

**Example: Sample jmxcli Invocation (using -m and defining multiple metrics from multiple Mbeans in one jmxcli session**

```
$ ./emctl jmxcli -t WebLogic -l "service:jmx:t3://host1:22048/jndi/
weblogic.management.mbeanservers.runtime" -u weblogic -c password -m
"com.bea:ApplicationRuntime=soa-infra,WebAppComponentRuntime=soa_server1_/b2b,*"

oracleHome=/ade/sparmesw_egcli/oracle/work/middleware/oms
userJars=
```

```
Connecting to server: service:jmx:t3://host1:22048/jndi/
weblogic.management.mbeanservers.runtime
Connecting as user: weblogic
Obtained 8 MBeans matching pattern com.bea:ApplicationRuntime=soa-
infra,WebAppComponentRuntime=soa_server1_/b2b,*.

Enter the target type for this metric: [myJ2EEApp] myCustomWLApp

Enter the target version: [1.0]

Enter the target metadata file: [./metadata/myCustomWLApp.xml]

Enter the default collections file: [./default_collection/myCustomWLApp.xml]
The file ./metadata/myCustomWLApp.xml already exists.

Do you want to overwrite the existing file, append to it, or quit <o/a/q>? [a]
```

*Note*: *Because the file already exists, it will be appended*.

```
Appending to existing file: ./metadata/myCustomWLApp.xml.
The available targets are:
0: (com.bea:ApplicationRuntime=soa-
infra,Name=JspServlet,ServerRuntime=soa_server1,Type=ServletRuntime,WebAppCompone
ntRuntime=soa_server1_/b2b)
1: (com.bea:ApplicationRuntime=soa-
infra,Name=transportServlet,ServerRuntime=soa_server1,Type=ServletRuntime,WebAppC
omponentRuntime=soa_server1_/b2b)
2: (com.bea:ApplicationRuntime=soa-
infra,Name=transportServletV,ServerRuntime=soa_server1,Type=ServletRuntime,WebApp
ComponentRuntime=soa_server1_/b2b)
3: (com.bea:ApplicationRuntime=soa-
infra,Name=b2b_starter_wls,ServerRuntime=soa_server1,Type=ServletRuntime,WebAppCo
mponentRuntime=soa_server1_/b2b)
4: (com.bea:ApplicationRuntime=soa-infra,Name=soa_server1_soa_server1_/
b2b,ServerRuntime=soa_server1,Type=PageFlowsRuntime,WebAppComponentRuntime=soa_se
rver1_/b2b)
5 (com.bea:ApplicationRuntime=soa-
infra,Name=WebServiceServlet,ServerRuntime=soa_server1,Type=ServletRuntime,WebApp
ComponentRuntime=soa_server1_/b2b)
6: (com.bea:ApplicationRuntime=soa-
infra,Name=RedirectUIServlet,ServerRuntime=soa_server1,Type=ServletRuntime,WebApp
ComponentRuntime=soa_server1_/b2b)
7: (com.bea:ApplicationRuntime=soa-
infra,Name=FileServlet,ServerRuntime=soa_server1,Type=ServletRuntime,WebAppCompon
entRuntime=soa_server1_/b2b)
Enter the index of target/MBean you wish to monitor or press <Ctrl-C> to quit: 4
Following metric source types are available for selected target(s):
        0: JMX Attributes
        1: JMX Operations
Enter the index of your choice or press <Ctrl-C> to quit: 0

Attributes are:
        0: AppName       Return Value: java.lang.String
        1: ContextPath   Return Value: java.lang.String
        2: HttpServerName        Return Value: java.lang.String
        3: Name  Return Value: java.lang.String
        4: PageFlows     Return Value: [Ljavax.management.ObjectName;
        5: Parent        Return Value: javax.management.ObjectName
        6: ServerName    Return Value: java.lang.String
        7: Type  Return Value: java.lang.String
```

```
Select one or more items as comma separated indices: 3,0,1

Number of possible columns in the resultant metric are 3.
Enter the name for this metric column at index=0 : [Name]
Is this column a KEY Column <y/n>? [n] y
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [Name]
Enter the NLSID for column: [Name]
Enter the UNIT for column "Name": [millisec, kb etc.. ]

Enter the name for this metric column at index=1 : [AppName]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [AppName]
Enter the NLSID for column: [AppName]
Enter the UNIT for column "AppName": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n]

Enter the name for this metric column at index=2 : [ContextPath]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [ContextPath]
Enter the NLSID for column: [ContextPath]
Enter the UNIT for column "ContextPath": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n]
Enter the name of this metric: PageFlowsRuntime
Enter the label for this metric: [PageFlowsRuntime]

Do you want periodic collection for this metric <y/n>? [n] y
Enter the collection interval in seconds: 3600
Periodic collection interval is: 3600 seconds.

Do you want to create another metric <y/n>? [n] y
```

This indicates more metrics need to be created in this `jmxcli` session. This process will repeat until you answer "n" to the question.

```
Do you want to create another metric <y/n>? [n]
Written the metadata xml file: ./metadata/myCustomWLApp.xml.
Updated the default collection file for myCustomWLApp at location ./
default_collection/myCustomWLApp.xml.
Exiting...
```

After the JMX command-line tool generates the target metadata and collection files, you can create the Oracle Plug-in archive (OPAR).

## Displaying Target Status Information

For the status information of your targets to appear correctly within the Enterprise Manager console, you must define a metric, called Response, that has a column, named Status, with a critical threshold set. The status of target instances of this type appears in the console as "Up" (available) if the metric value is below the critical threshold. When the threshold is exceeded, the target status appears as "Down" in the console.

You can create the Response metric in another `jmxcli` session (append the metric to the metadata and collection files created in an earlier session).

**Example: Adding a Response Metric**

```
setenv USER_JARS $T_WORK/middleware/wlserver_10.3/server/lib/weblogic.jar
```

This is required as some MBeans return WebLogic-specific classes which the JMX client (`jmxcli`) needs in its classpath.

```
$ ./emctl jmxcli -t WebLogic -l "service:jmx:t3://host1:22048/jndi/
weblogic.management.mbeanservers.runtime" -u weblogic -c password -m
com.bea:Type=ApplicationRuntime,Name=soa-infra,*"
```

For J2EE applications deployed on WebLogic it may be appropriate to make the ActiveVersionState JMX attribute of the ApplicationRuntime Mbean corresponding to the application deployment as the Status column. However, any other attribute of any other relevant Mbean to the application could also be used.

```
oracleHome=/ade/sparmesw_egcli/oracle/work/middleware/oms
userJars=
Connecting to server: service:jmx:t3://host1:22048/jndi/
weblogic.management.mbeanservers.runtime
Connecting as user: weblogic
Obtained 1 MBeans matching pattern
 com.bea:Type=ApplicationRuntime,Name=soa-infra,*.
Enter the target type for this metric: [myJ2EEApp] myCustomWLApp
Enter the target version: [1.0]
Enter the target metadata file: [./metadata/myCustomWLApp.xml]
Enter the default collections file: [./default_collection/myCustomWLApp.xml]
The file ./metadata/myCustomWLApp.xml already exists.

Do you want to overwrite the existing file, append to it, or quit <o/a/q>? [a]
Appending to existing file: ./metadata/myCustomWLApp.xml.
The available targets are:
0: (com.bea:Name=soa-infra,ServerRuntime=soa_server1,Type=ApplicationRuntime)
Enter the index of target/MBean you wish to monitor or press <Ctrl-C> to quit: 0
Following metric source types are available for selected target(s):
        0: JMX Attributes
        1: JMX Operations
Enter the index of your choice or press <Ctrl-C> to quit: 0
Attributes are:
        0: ActiveVersionState     Return Value: java.lang.Integer
        1: ApplicationName        Return Value: java.lang.String
        2: ApplicationVersion     Return Value: java.lang.String
        3: ClassRedefinitionRuntime      Return Value:
javax.management.ObjectName
        4: ComponentRuntimes      Return Value: [Ljavax.management.ObjectName;
        5: EAR    Return Value: java.lang.Boolean
        6: HealthState    Return Value: weblogic.health.HealthState
        7: KodoPersistenceUnitRuntimes   Return Value:
[Ljavax.management.ObjectName;
        8: LibraryRuntimes        Return Value: [Ljavax.management.ObjectName;
        9: MaxThreadsConstraintRuntimes  Return Value:
[Ljavax.management.ObjectName;
        10: MinThreadsConstraintRuntimes        Return Value:
[Ljavax.management.ObjectName;
        11: Name         Return Value: java.lang.String
        12: OptionalPackageRuntimes      Return Value:
[Ljavax.management.ObjectName;
        13: Parent       Return Value: javax.management.ObjectName
        14: QueryCacheRuntimes    Return Value: [Ljavax.management.ObjectName;
        15: RequestClassRuntimes        Return Value:
[Ljavax.management.ObjectName;
```

```
    16: Type            Return Value: java.lang.String
    17: WorkManagerRuntimes  Return Value: [Ljavax.management.ObjectName;
    18: WseeRuntimes        Return Value: [Ljavax.management.ObjectName;

Select one or more items as comma separated indices: 0

Number of possible columns in the resultant metric are 1.

Enter the name for this metric column at index=0 : [ActiveVersionState] Status
```

*Note: The column name must be "Status".*

In this case since the Enterprise Manager target is an application. You must mark
the target as *down* even when the *container* is down, i.e., when the JMX connection
cannot be established. To achieve this (show the application as down when the server
is down as well) you can add the following *QueryDescriptor* property to the Response
metric generated in the metadata file:

```
<Property NAME="valueWhenDown" SCOPE="GLOBAL">0</Property>
```

Here, 0 is the value returned for the Status column when the JMX connection cannot
be established with the server (the server itself is down).

```
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [Status]
Enter the NLSID for column: [Status]
Enter the UNIT for column "Status": [millisec, kb etc.. ]
Do you want to create a threshold for this column <y/n>? [n] y
Creating threshold!!
Following operators are available for creating thresholds:
 0: GT
 1: EQ
 2: LT
 3: LE
 4: GE
 5: CONTAINS
 6: NE
 7: MATCH

Enter the index of your choice or press <Ctrl-C> to quit: 6
Enter the CRITICAL threshold: [NotDefined] 2
```

Status of target is marked down if a CRITICAL THRESHOLD is triggered on the Status
column of the Response Metric. In this case if value != ACTIVATED (such as: != 2)

```
Enter the WARNING threshold: [NotDefined]
Enter the number of occurrences that trigger threshold: [6] 1
Enter the message to be used when threshold is triggered: [Status is %value%
and has crossed warning (%warning_threshold%) or critical (%critical_threshold%)
threshold.]
Enter NLSID for the message used when threshold is triggered: [Status_cond]

Enter the name of this metric: Response
```

*Note: The metric name must be "Response".*

```
Enter the label for this metric: [Response]

Do you want periodic collection for this metric <y/n>? [n] y
```

```
Enter the collection interval in seconds: 30
Periodic collection interval is: 30 seconds.

Do you want to create another metric <y/n>? [n]
Written the metadata xml file: ./metadata/myCustomWLApp.xml.
Updated the default collection file for myCustomWLApp at location ./
default_collection/myCustomWLApp.xml.
Exiting...
```

## Using the Metadata and Default Collection Files

Look at the `currentDir/metadata` and `currentDir/default_collection` directories to see the `myTarget.xml` files (for the target type you specified earlier).

You can use these files as follows:

- Convert the files to an Oracle Plug-in Archive (OPAR). See Creating the Plug-in Archive.
- Move the OPAR to the OMS. See Importing and Deploying the Plug-in Archive into Enterprise Manager.
- Push the OPAR to the Agents. See Importing and Deploying the Plug-in Archive into Enterprise Manager.
- Create custom target instances. See Adding a Target Instance for a Custom J2EE Application on WebLogic

If you want the status information of your targets to appear correctly in the Enterprise Manager console, you need to define a Response metric. See Displaying Target Status Information for more information.

# Adding a Target to a Management Agent

Once the plug-in has been deployed to the OMS, you are ready to add targets defined by your metadata plug-in to different monitoring Management Agents.

For illustrative purposes, the following steps show how to add the sample CalculatorService and TrafficLight as targets.

## Adding a Web Services Target - CalculatorService

To add the CalculatorService target, perform the following steps:

1. From the **Setup** menu, select **Add Target** and then **Add Targets Manaully**.
2. Select **Add Non-Host Targets by Specifying Target Monitoring Properties**.
3. From the **Target Type** menu, select **CalculatorService**.
4. From the **Monitoring Agent** menu, select the required monitoring Management Agent.
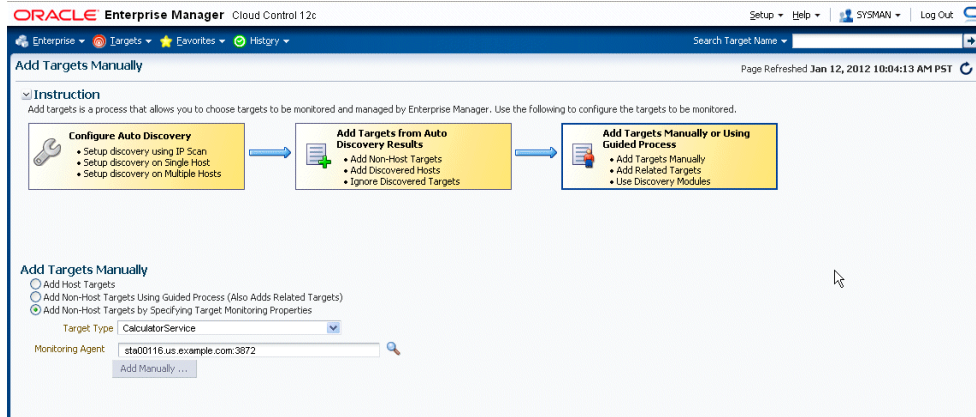
5. Click **Add Manaully** to proceed.

6. Enter the property values of the target to be monitored.



7. Click **OK** to complete the process. The confirmation window displays information on the newly added target.

## Adding a WS-Management Target - TrafficLight

To add the sample TrafficLight target, perform the following steps:

1. From the **Setup** menu, choose **Add Target** and then **Add Targets Manually**.

2. Select **Add Non-Host Targets by Specifying Target Monitoring Properties**.

3. Select **TrafficLight** from the **Target Type** drop-down menu.

4. Select the desired agent from the **Monitoring Agent** drop-down menu.

**5.** Click **Add Manually** to proceed.

**6.** Enter the property values of the target to be monitored.



**7.** Click **OK** to complete the process. The confirmation window displays information on the newly added target.

## Configuring a Standalone Java Application or JVM Target

If you deployed a plug-in that defines a standalone Java application or you want to use the built-in JVM target type, you can begin configuring your JVM or JMX-enabled Java application targets so that metrics for these targets can be collected in Enterprise Manager Cloud Control.

On the system running the JVM, install an Enterprise Manager Agent version 10.2.0.3 or later. Although recommended, this is not necessary for JVM and standalone Java

application targets: The monitoring Agent does NOT have to be local to the target JVM.

To add the JVM target instance, perform the following steps:

1. From the **Setup** menu of Enterprise Manager console (top right), select **Add Target** and then **Add Targets Manually.**

2. Select **Add Non-Host Targets by Specifying Target Monitoring Properties**.

3. From the **Target Type** menu, select **JVM**.

4. From the **Monitoring Agent** list, select the required Management Agent (preferably a Management Agent local to the JVM being monitored).
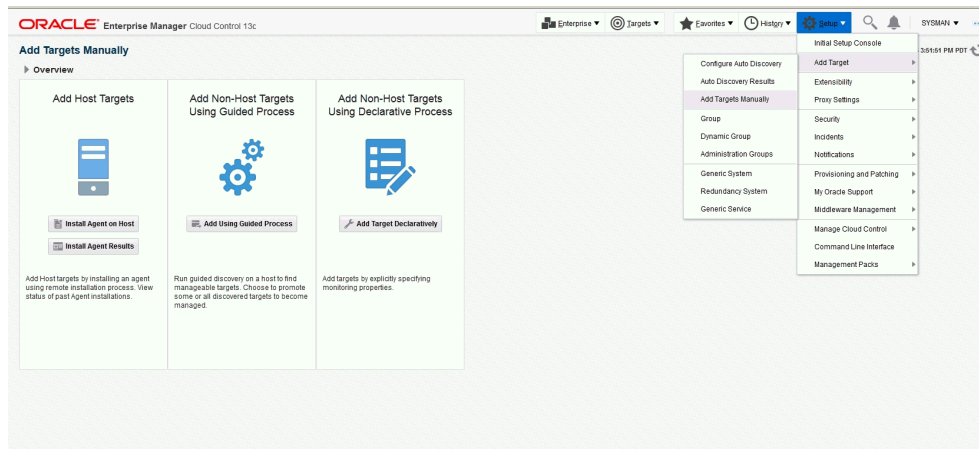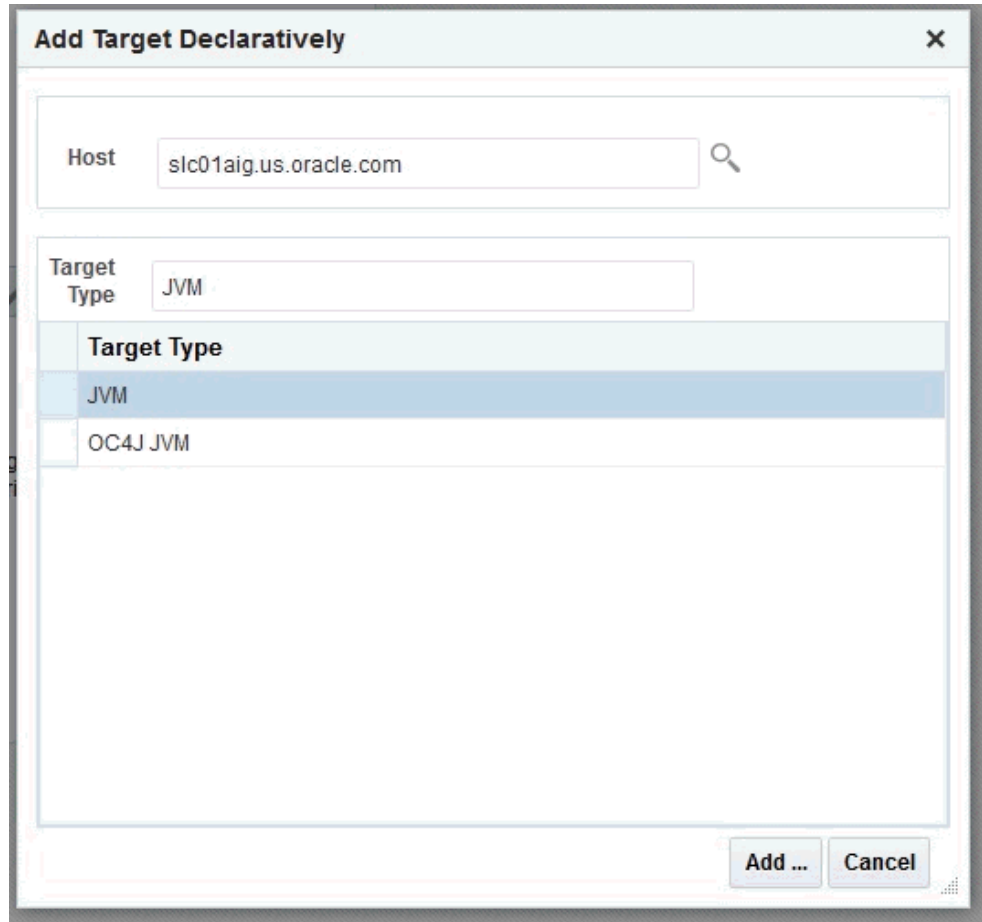
5. Enter the instance properties for this JVM or Java application instance that the Management Agent needs to monitor, then click **OK**.
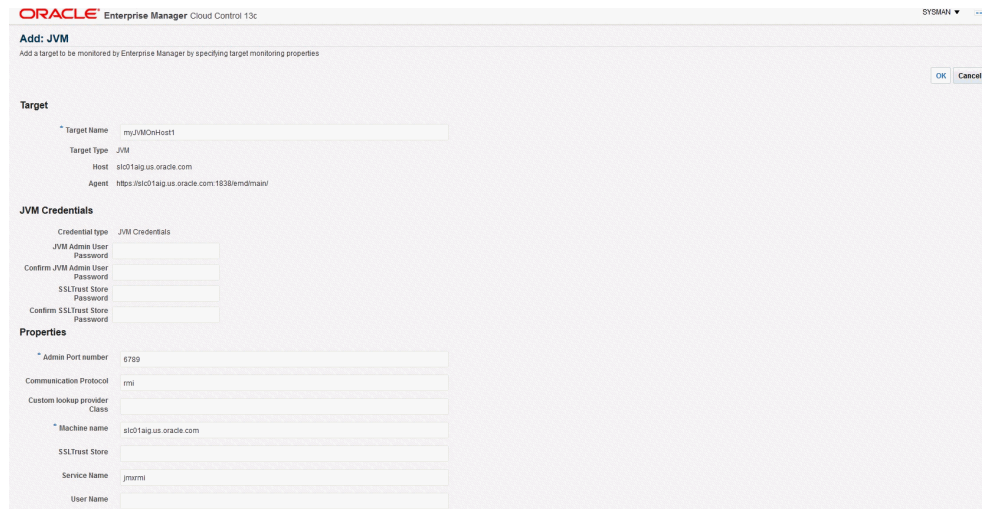


Table 18-1 provides definitions for the instance properties.

**Table 18-1    JVM Instance Properties**

| Property | Definition |
|---|---|
| **Name** | Target name for this JVM instance. |
| **MachineName** | Host name where this JVM is running. |
| **Admin Port Number** | Port number a JSR-160 client can use (such as jconsole when using the "remote" option) to connect to the JVM. (This is the port specified for the `-Dcom.sun.management.jmxremote.port` property when the JVM is started up to enable remote management.) |
| **User Name** | Required if JVM started with: `Dcom.sun.management.jmxremote.authenticate=true` with a password and access file. |
| **JVM Admin User Password** | See the preceding User Name property. |
| **Communication Protocol** | Establishes a connection to the MBeanServer on the target JVM. This corresponds to the properties of the JMX ServiceURL needed to establish the JMX connection to the target MBeanServer. The default of rmi should be retained. |
| **Service Name** | Establishes a connection to the MBeanServer on the target JVM. This corresponds to the properties of the JMX ServiceURL needed to establish the JMX connection to the target MBeanServer. The default of jmxrmi should be kept. |
| **SSL Trust Store** | Location of the SSL Trust Store, which is needed if the target JVM has SSL enabled with `com.sun.management.jmxremote.ssl=true` on its startup. |
| **SSL Trust Store Password** | Password needed to access the SSL Trust Store path. |
| **Custom Lookup Provider Class** | Full package name of a user-implemented Java lookup class that can be integrated into the Enterprise Manager client and be used to perform a custom lookup of the MBeanServer through LDAP or other lookup protocols. |

6. Navigate to the All Metrics page of the added JVM (Java application) target to see the metrics collected from the JVM (Java application) to Enterprise Manager. These metrics are exposed by the platform MBeans, which is available on JDK1.5 or above, or from application-defined MBeans for your Java application.

To navigate to JVM target home page from the **Targets** menu, choose **All Targets** and then select your JVM target instance.

To navigate to the **All Metrics** page, from the **Target** menu, select **Monitoring** and then **All Metrics** from the JVM target's home page menu.
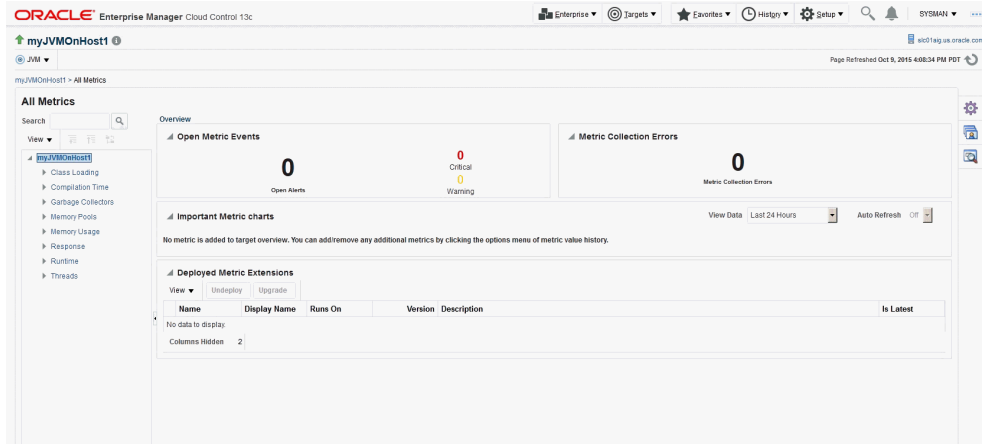
The following graphic shows the collected metric details.
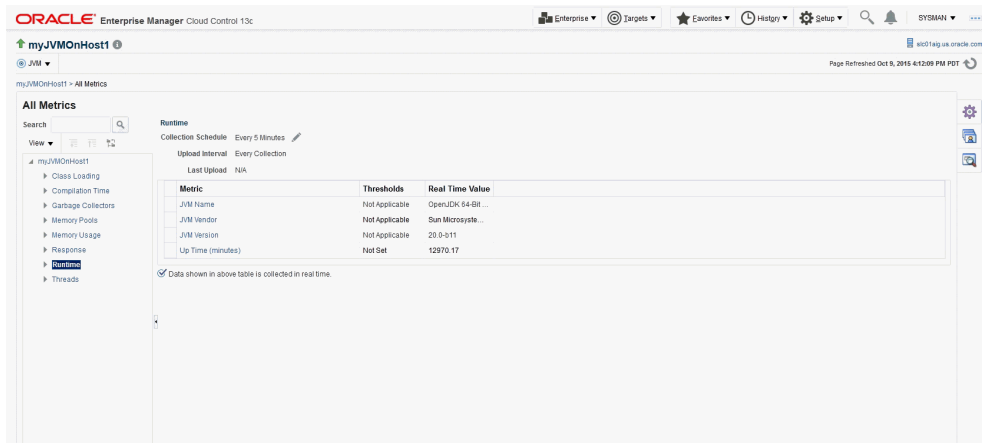


**Table 18-2    Properties the Fetchlet Uses**

| Property | Default | Description |
|---|---|---|
| MachineName | localhost | MBean server host machine name. |
| Port | 8888 | Port on which the MBean server is listening for connections. |
| Username | null | User name if required for a connection. |
| Password | null | Password if required for a connection. |
| Protocol | rmi | Protocol used for the connection. |
| Service | jmxrmi | Service used for the connection. |
| SSLTrustStore | null | Path to the SSLTrustStore. |
| SSLTrustStorePassword | null | Password needed to access the SSLTrustStore path. |

# Adding a Target Instance for a Custom J2EE Application on WebLogic

You have a custom J2EE application on WebLogic from which you need to collect custom metrics into Enterprise Manager that are exposed via JMX Mbeans. Once you have defined and deployed a plug-in that defines your custom target type, you can begin configuring your JMX-enabled J2EE application target instances on the various Management Agents to where you deployed the plug-ins. This enables Enterprise Manager to collect metrics for these target instances.

1. From the **Setup** menu, select **Add Target** and then **Add Target Manually**. Select the **Add non-Host targets by specifying Target Monitoring Properties** option.

2. Select your custom target type created earlier and deployed to the OMS

3. Select the monitoring Management Agent where you want to create an instance of this target type (this should preferably be an emagent local to the target)
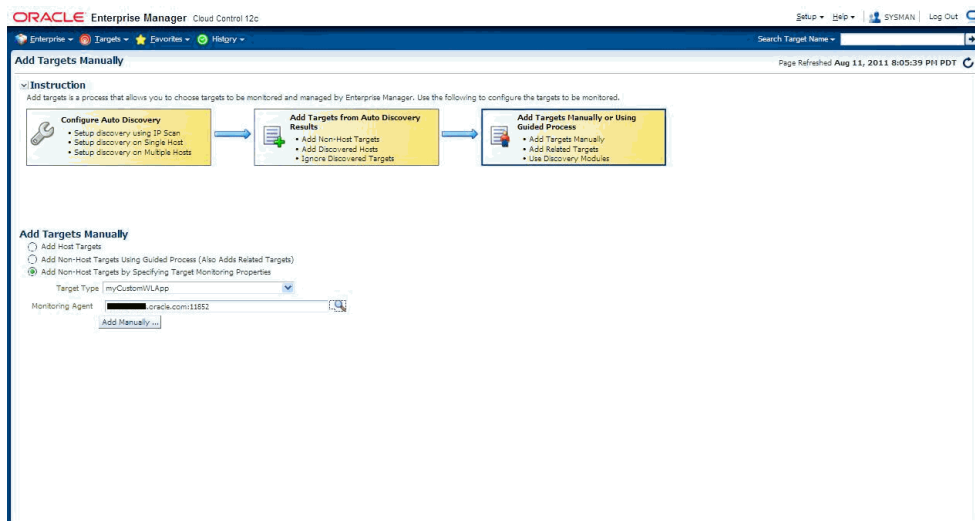


4. Click **Add Manually**.

5. Enter the requisite target properties, as shown in the following graphic, then click **OK**. The newly added target appears in the "All Targets" list.

> ✎ **Note:**
>
> If the jmxcli was invoked with a *-l <serviceURL>* option, then the following screen would prompt for a *JMXServiceURL* property which can be entered as follows:
>
> ```
> service:jmx:t3://<host>:<port>/jndi/
> weblogic.management.mbeanservers.runtime
> ```
>
> Instead of entering the *MachineName*, *Admin Port* number and *Service Name* as three distinct target instance properties.

**Table 18-3    Target Properties**

| Property | Definition |
|---|---|
| **Name** | Unique name for this target instance. |
| **MachineName** | Host name/IP Address of the system running the 9.x version or later of the Oracle WebLogic Application Server. |
| **Username** | User Name used to establish the JMX connection to the WebLogic server. This could be either an administrator or monitor user. |
| **JVM Admin User Password** | Password for preceding user. |
| **Communication Protocol** | t3 (default) or t3s. |
| **Service Name** | *weblogic.management.mbeanservers.runtime* (or other MbeanServer where the application registers its Mbeans). |
| **Metric Source** | WebLogic |

The metrics created can be viewed by navigating to the target instance home page and navigating to the **All Metrics** page (from the **Target** menu, choose **Monitoring** and then **All Metrics**).

# Monitoring Credential Setup

Some target types require monitoring credentials to be set for target instances. In the demo plug-ins, both CalculatorService and TrafficLight require monitoring credentials. The following steps demonstrate how to set up the credentials:

1. From the **Setup** menu, select **Security** and then **Monitoring Credentials**.

2. Select **CalculatorService** and then click **Manage Monitoring Credentials**.

3. Select **Calculator1** and then click **Set Credentials**.



4. Select **AliasCredential** from **Credential Type**. Enter values for **Alias** and **Password**.

5. Click **Save** to finish.

6. Repeat the above steps for the target **TrafficLight1**.

# Viewing Monitored Metrics

With a target instance added to the Management Agent for monitoring, you can now view metrics defined for your target type. As before, the sample targets are used to illustrate the procedure.

1. From the **All Targets** page, click the target you added in the previous step. Enterprise Manager takes you to that target's home page.

2. From the **Target** menu, select **Monitoring** and then **All Metrics**. The **All Metrics** page appears for the monitored target. An expandable tree list for each metric enables you to drill down to view specific metric parameters, as shown below:

# Creating JMX Metric Extensions

If you wish to collect metrics from your custom J2EE application deployed on Oracle Fusion middleware and exposed via JMX attributes into Enterprise Manager, you can use either the Enterprise Manager console or the `jmxcli` command line tool. The latter also supports defining Metric Extensions from JMX operations and supports the creation of a Metric Extension Archive (MEA) which then must be imported into the OMS via the console and then tested and deployed to the desired J2EE application target instances representing your custom application.

> **Note:**
>
> While you can select attributes that are not open types using the Mbean browser, the JMX metric extension UI supports only open type attributes. An error will occur if the UI is used to create metrics by selecting attributes which are not open types.

## Using the Enterprise Manager Console

1. From the **Enterprise** menu, select **Monitoring** and the **Metric Extensions**. The Metric Extension page displays.

2. Click **Create** to create a Metric Extension.



3. Select "Application Deployment" target type (or any other appropriate Enterprise Manager target type for which this metric needs to be defined) and specify a meaningful name for your metric extension. Keep in mind that you might eventually end up creating additional metric extensions on the "Application Deployment" target type both for this application and for other custom applications so it is desirable to capture both the metric name and the application name in the metric extension name, whenever possible.

   Also select JMX for the Adapter.

   Note the "Collection Schedule" section below the "General Properties" section. This is where you define how often this metric is to be collected, or if this is realtime-only metric (in which case the **Disabled** button should be selected.).

If "Alerting and Historical Trending" is selected, you can also select an **Upload Interval**, which indicates which samples (whose frequency is specified in the "Repeat Every" field) are uploaded to the Enterprise Manager repository for historical trending. For example if Collection frequency is specified as 15 minutes and the Upload Interval is 3, then every 3rd sample will be uploaded into the repository (every 45 minutes) and will be available for historical trending. However "alerts" that are possibly triggered due to threshold violations will be available for every collection (15 minutes).



4. Click **Next** and specify the required properties needed for a JMX-based metric. These are defined in the **Basic Properties** section and are:

   • Metric: The Mbean ObjectName or Pattern and

   • Column Order: A semi-colon separated list of JMX attributes for above Mbean (if a metric needs to be defined using a JMX operation, use the `jmxcli` as shown in a following section)

   Note that the Mbean ObjectName or pattern defined previously must not have any server-specific key properties defined. These properties may be replaced with a wildcard ("*").

   For example, if an Mbean object name is *com.bea:Type=foo,Location=Server1,Name=abc* then it may be appropriate to define this as *com.bea:Type=foo,\** in the "Metric" property described above. Also, if the Mbean ObjectName is a pattern, please be aware that multiple Mbeans could be returned making this metric a "table" with multiple rows (each row representing the JMX attributes of an Mbean matching the ObjectName pattern). In this case we need to define at least one or more columns as Key columns so that each row is unique in the resultant metric.
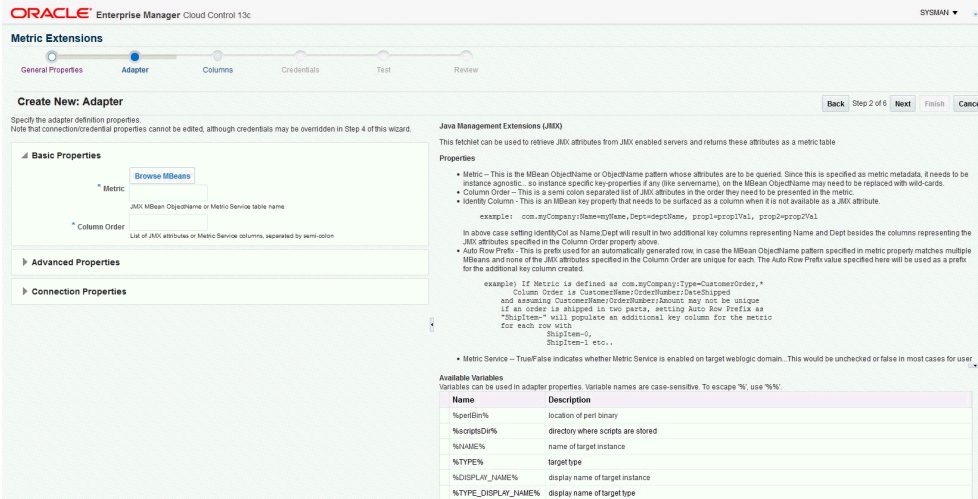
For the preceding step, there is a **Browse Mbeans** button that makes it easier to configure these two properties by allowing you to browse an MbeanServer and selecting an Mbean and its JMX attributes that need to be represented by this metric being defined in the metric extension.

If you click **Browse Mbeans**, you must perform the following in sequence.

- Select the Target: Select an instance of the target type that you need to use to define this metric. This target instance is used to help configure the metric and does not have to be the target instance on which the metric is eventually defined.

- Enter the Mbean Pattern: Here, you enter an Mbean Object Name or pattern for the Mbean you are interested in monitoring

- Click **List Mbeans**: This will be displayed in the table under "Select Mbean and Attributes", the Mbeans that match the Mbean pattern or the text "No Mbean listed or none matches Mbean Pattern" if there is no match. You can iteratively update the previous "Enter Mbean pattern" field and click **List Mbeans** to refine the list of Mbeans displayed.
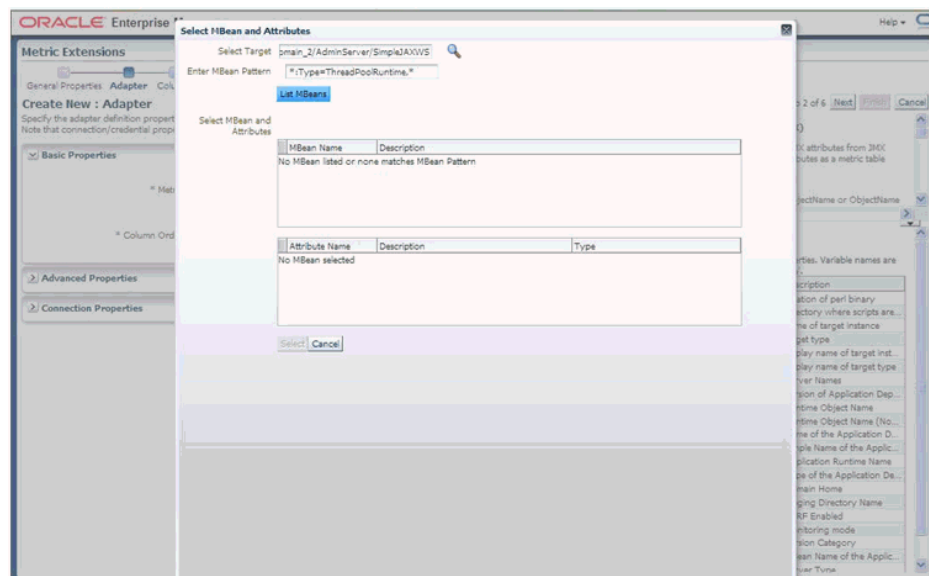
- Select an Mbean of interest: This will automatically populate the table below with the JMX attributes for the selected Mbean.
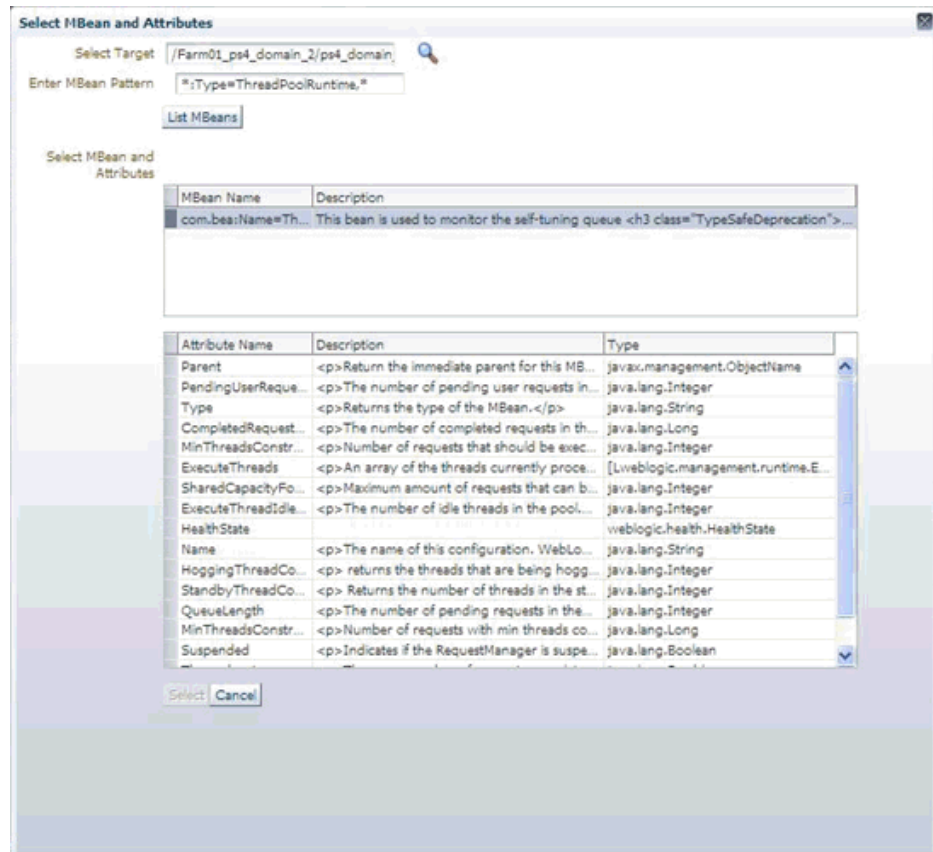


You can multi-select (using Control + click) multiple attributes and then click **Select** to accept the selections.
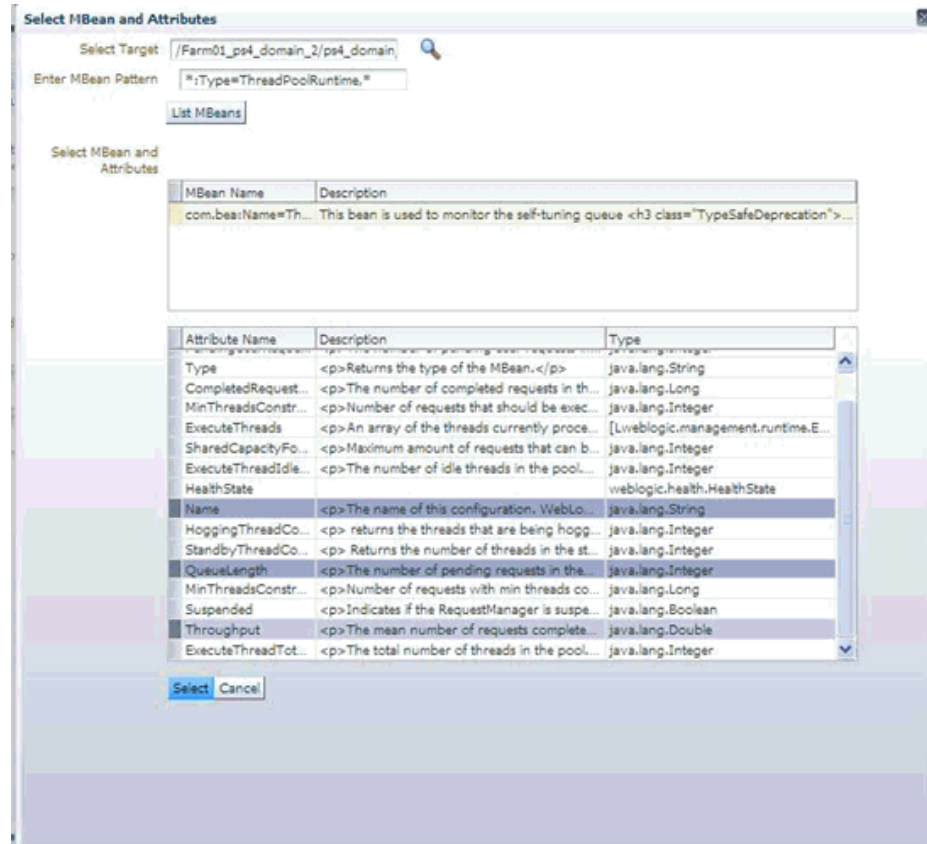
5. You must now specify the required parameters "Metric" and "Column Order" needed to define a JMX based metric extension.

   Note that the Mbean name populated in the "Metric" field should not have any instance specific information in its key properties (like Location=Server1 or ServerName=foo) if this metric extension can be applied to multiple servers besides the one that was selected/used to configure the metric extension using the "Browse Mbean" wizard above. These instance-specific key properties could be replaced with a wildcard "*" as appropriate to make this a valid Mbean ObjectName pattern.
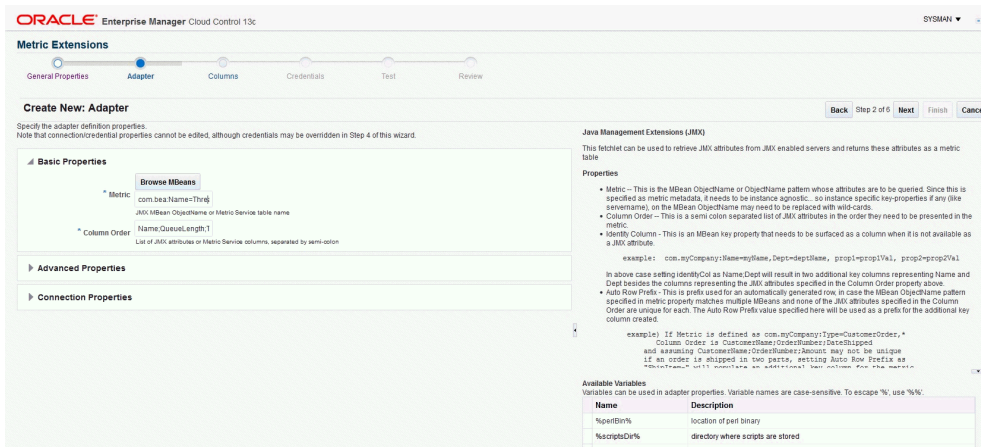


**Explanation of Specifiable Properties**

Required Properties:

- metric -- This is the MBean ObjectName or ObjectName pattern whose attributes are to be queried. Since this is specified as metric metadata, it needs to be instance agnostic so instance specific key-properties if any (like servername), on the MBean ObjectName may need to be replaced with wildcards.

- columnOrder -- This is a semi colon separated list of JMX attributes in the order they need to be presented in the metric

Advance Properties:

- identityCol -- This is an MBean key property that needs to be surfaced as a column when it is not available as a JMX attribute. For example: com.myCompany:Name=myName,Dept=deptName, prop1=prop1Val, prop2=prop2Val In above case setting identityCol as Name;Dept (note that separator is a semi-colon) will result in two additional key columns representing Name and Dept besides the columns representing the JMX attributes specified in the columnOrder property above.

- autoRowId -- This is the prefix used for an automatically generated row in case the MBean ObjectName pattern specified in metric property matches multiple MBeans and none of the JMX attributes specified in the columnOrder are unique for each. The autoRowId value specified here will be used as a prefix for the additional key column created. For example, if the metric is defined as com.myCompany:Type=CustomerOrder,* columnOrder is CustomerName;OrderNumber;DateShipped (and assuming customerName;OrderNumber;DateShipped may not be unique if an order is shipped in two parts).

  Setting autoRowId as "ShipItem-" will populate an additional key column for the metric for each row with ShipItem-0, ShipItem-1.

- MetricService -- True/False indicates whether MetricService is enabled on the target WebLogic domain. This would be unchecked or false in most cases for user-defined metrics except when metrics that are exposed via the Oracle DMS MBean needs to be collected. If set to true, then the basic property "metric" above should represent the MetricService table name and the basic property "columnOrder" will represent a semicolon separated list of column names for aforementioned MetricService table.

6. Specify the Columns for this metric (if you have used the "Browse Mbeans" step earlier, then these columns are automatically pre-filled for you). You may need to edit these pre-created columns by the "Browse Mbean" wizard to specify columns that are "Key columns". This done in the event an Mbean pattern is specified in the previous step for the "Metric" property, and multiple Mbeans could match this Mbean pattern for any of the target instances to which this metric extension will be applied to.

If the order of the columns are changed (using Move Up - Move Down buttons) then the corresponding order of the semi-colon separated columns in the "Column Order" property in the previous step also needs to be updated accordingly (using **Back**).
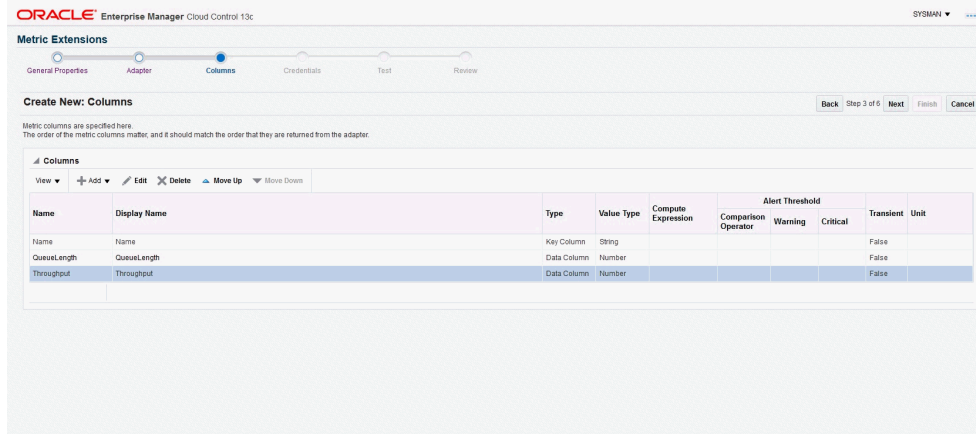
If needed, edit the columns as desired to make them a Key Column as shown in the following graphic.
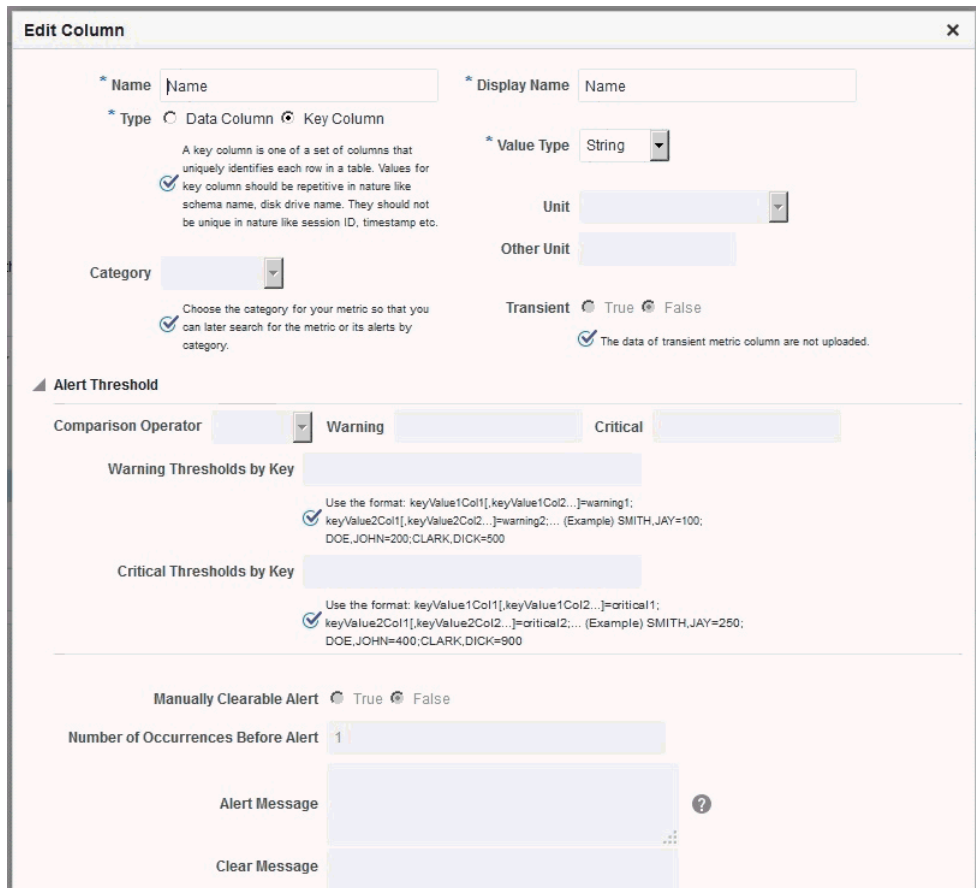


Once columns are labeled and edited, click **Next**. We are now ready to test the metric extension

7. Click **Add** to select a target instance on which to test this metric extension. This could preferably be a different target instance than the one used to define the

metric extension (if the **Browse Mbeans** button was used to help in defining the metric extension earlier).
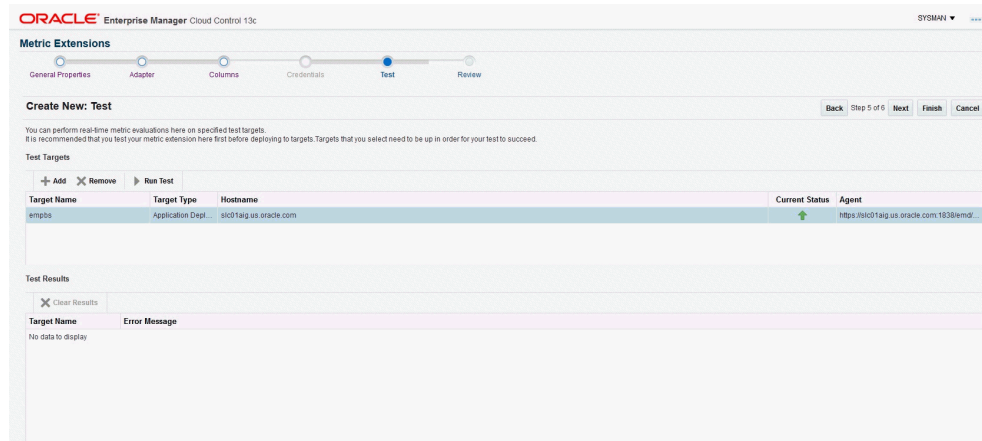


Now select a target instance in the Test Targets table and then click **Run Test** above that table.

The metric values are displayed in the Test Results table (if there are errors ,then those are also shown ).

If errors are present, click **Back** and fix the errors and re-run the test.
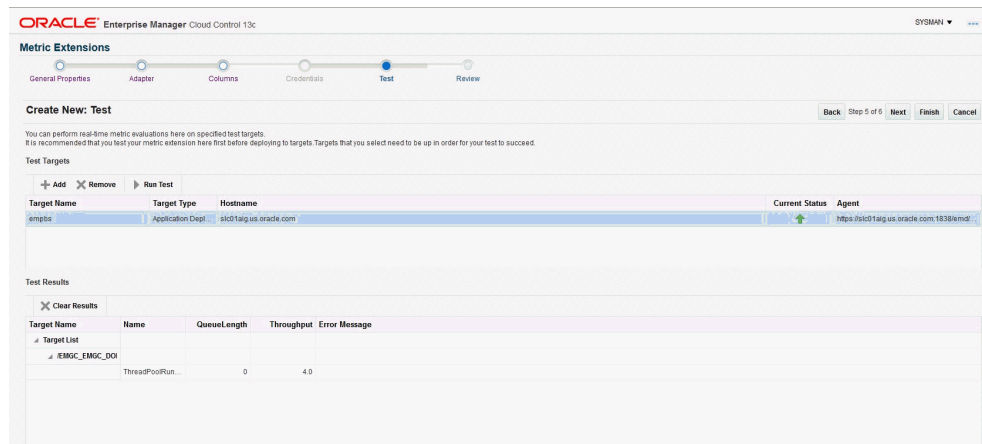


8. Once satisfied with the Test, click **Next** to view a summary of the metric extension and then click the **Finish** to define the metric extension.
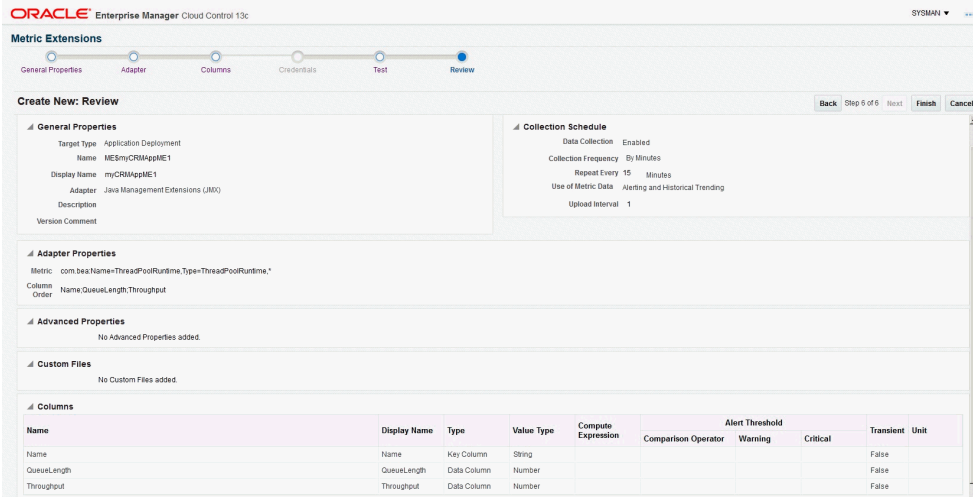
9. Before deploying the metric extension to selected target instances the metric extension needs to be saved as a "Deployable draft". This will let the metric extension designer deploy the metric extension to selected target instances and verify the metric collection but will prevent other administrators from deploying this metric extension until after it has been tested and the designer is satisfied.
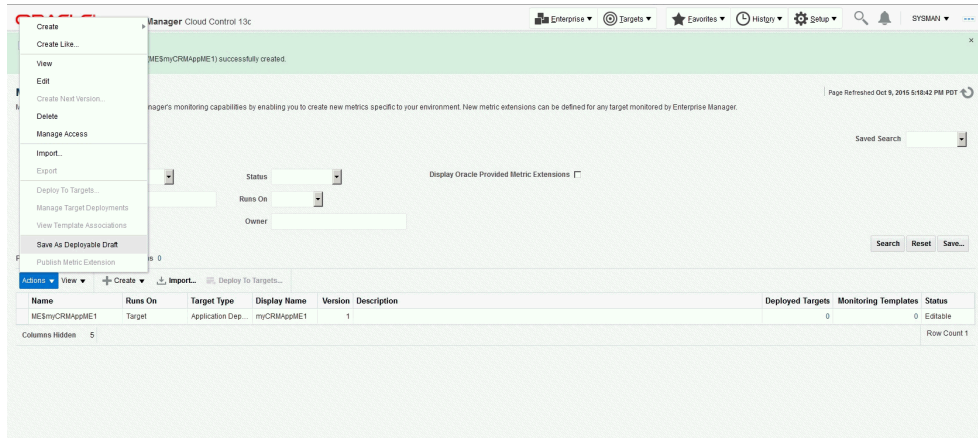


10. Select the metric extension just created and saved as a deployable draft. From the **Actions** menu, choose **Deploy to Targets**.
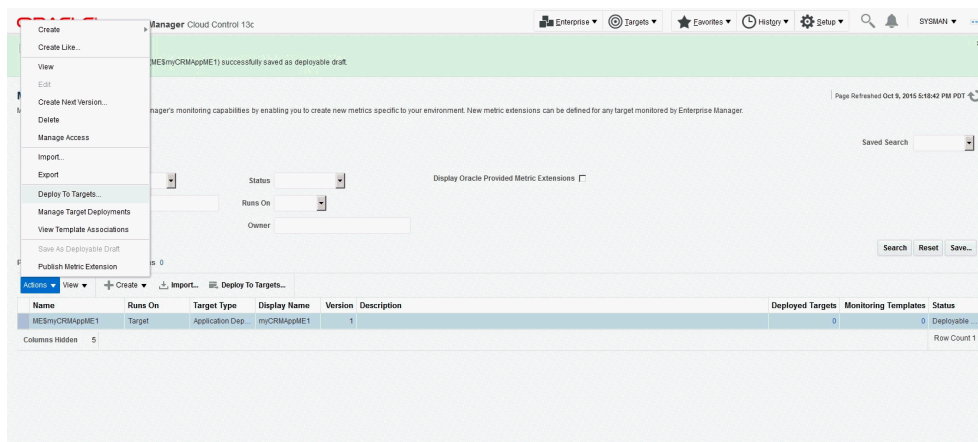
**11.** Select the target instances that this metric extension needs to be tested on and click **Submit**. For example, if the metric extension is defined on an "Application Deployment" target type and represents a metric from a Custom Mbean registered by a custom JEE application, the instances of that custom application could be selected. This will schedule a job to asynchronously deploy the metric extension to the Management Agents monitoring the selected targets.



**12.** Monitor the status of the Pending deploy operation of the metric extension to selected targets by refreshing this page periodically to monitor the Status column and Failed Deploy Operations table for any possible errors during deployment.
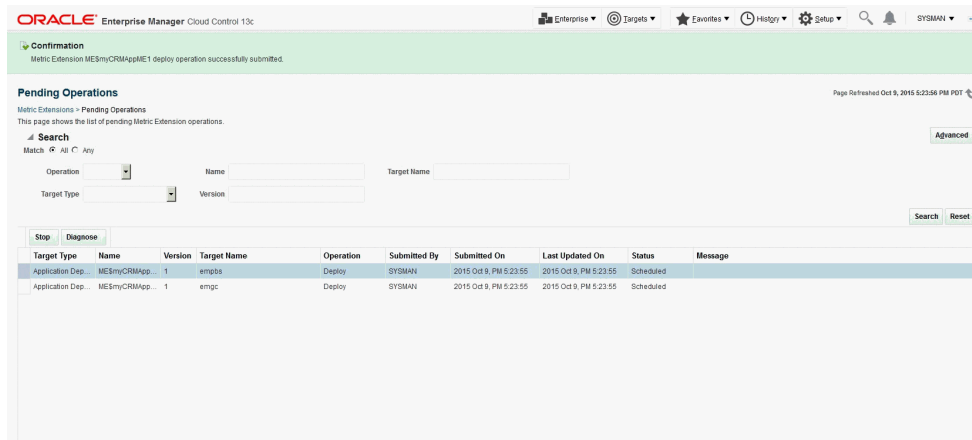


**13.** From the **Enterprise** menu, select **Monitoring** and then **Metric Extension**. On the Metric Extension home page, your metric extension appears as a row in the table with a column "Deployed Targets" representing the count of the number of targets this metric extension is deployed to.

From the **Actions** menu, choose **Manage Target Deployments** from the table after selecting the desired metric extension. This will list the target instances this metric extension is deployed to.
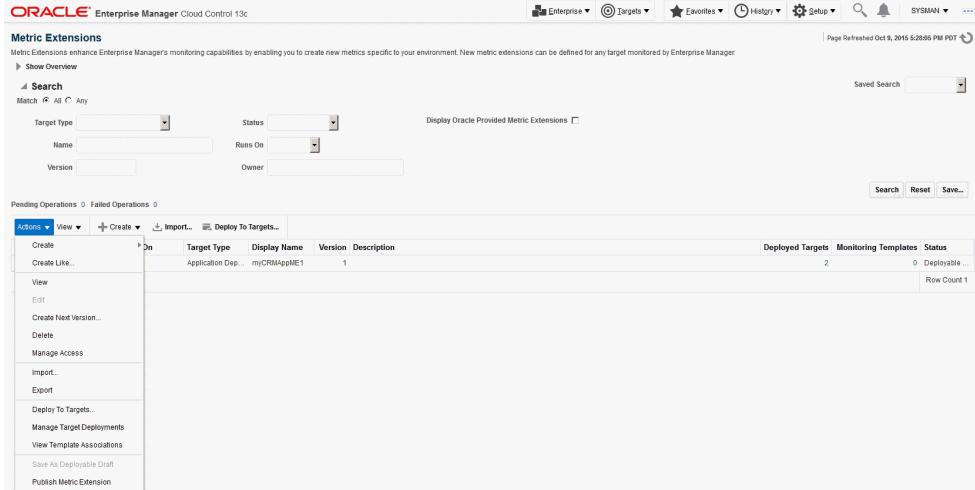
**14.** Click on the value in the "Target Name" column for the target instance you want to verify the metric extension on. This takes you to the home page of the target.
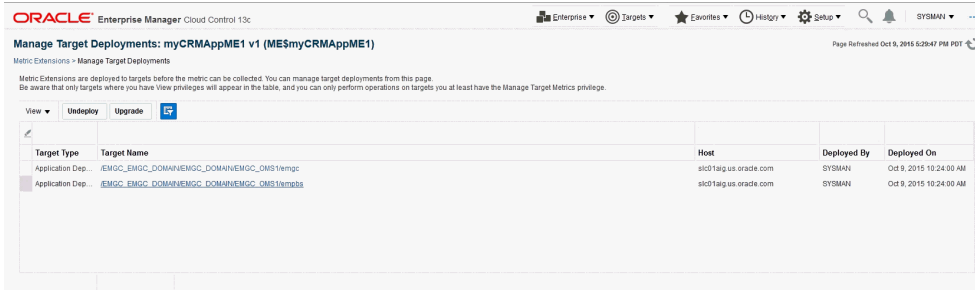


**15.** For middleware targets, navigate to the *Target Type*/Monitoring/ Performance Summary (or in general to the *Target Type* /Monitoring /All Metrics) page.
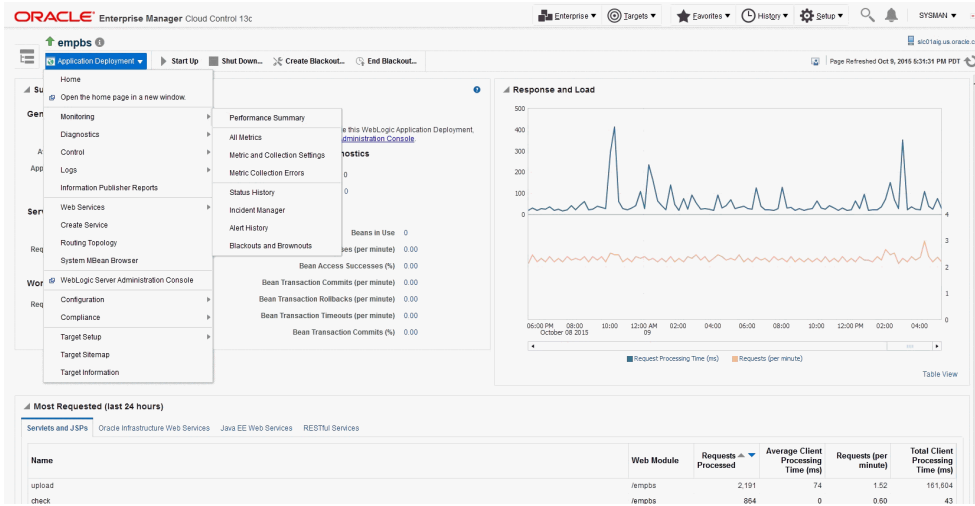
**16.** From the Performance Summary page, the newly created metric will be visible on the Metric Palette and can be selected and charted on the page.
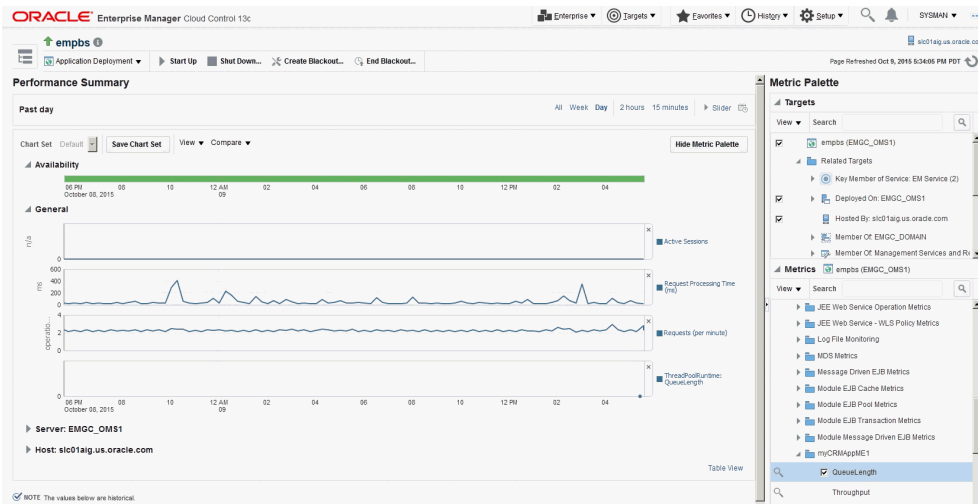


**17.** Once satisfied with testing the metric extension on one or more target instances, the metric extension can be published from the Metric Extension page (from the Actions menu, choose Publish Metric Extensions) and then deployed to remaining target instances.

# Using the JMXCLI to create a Metric Extension Archive

If you do not wish to use the Enterprise Manager console (or do not want to surface an Enterprise Manager metric exposed via a JMX operation), you can use the command line tool JMXCLI to create a Metric Extension Archive. This can then be imported into the OMS, edited, tested, published and then deployed to required instances of the target type on which it is defined. The following illustrates the use of jmxcli in creating a Metric Extension archive.

**1.** cd *Agent_Instance_Home*/bin

**2.** setenv USER_JARS $T_WORK/middleware/wlserver_10.3/server/lib/weblogic.jar (this should not be necessary if your Mbeans just return JMX Open Types and not any custom classes).

**3.** emctl jmxcli -t WebLogic -MEXT -l "service:jmx:t3://host1:7018/jndi/weblogic.management.mbeanservers.runtime" -u weblogic -c *password* -m "*:Type=ThreadPoolRuntime,*" -w /scratch/TEMP/

**Options**:

-l : JMX serviceURL to connect to the WebLogic server. Replace the host:port above with what is appropriate for your instance

-u : WebLogic user having access to required MBeans

-c : Password for the WebLogic user

-m : Mbean ObjectName or pattern.

-w : Temporary work directory where the Metric Extension Archive (which can later be imported into the OMS console) is created.

```
Oracle Enterprise Manager 12c Release 1 Cloud Control 12.1.0.0.0
Copyright (c) 1996, 2011 Oracle Corporation.  All rights reserved.
Using Plugin Root /ade/sparmesw_egc802/oracle/emagent/gcagent/
plugins/oracle.sysman.emas.agent.plugin_12.1.0.0.0
Connecting to server: service:jmx:t3://host1:7018/jndi/
weblogic.management.mbeanservers.runtimeConnecting as user: weblogic
Obtained 1 MBeans matching pattern *:Type=ThreadPoolRuntime,*.
Enter an existing target type for this Metric Extension:
[j2ee_application]
Enter the name of the Metric Extension: [myMEXT] myAppME_1
Enter the Metric Extension version: [1.0]
Enter the Metric Extension metadata file location: [./metadata/
ME#24#myAppME_1.xml]
Enter the Metric Extension collection file location: [./collection/
ME#24#myAppME_1.xml]
Enter a label for this Metric Extension: [myAppME_1]
Enter a description for this Metric Extension: [myAppME_1]
The available targets are:
0: This bean is used to monitor the self-tuning queue  <h3
class="TypeSafeDeprecation">Deprecation of MBeanHome and Type-Safe
Interfaces</h3>
<p class="TypeSafeDeprecation">This is a type-safe interface for a
WebLogic Server MBean,
which you can import into your client classes and access through
<code>weblogic.management.MBeanHome</code>.
As of 9.0, the <code>MBeanHome</code> interface and all type-safe
interfaces for WebLogic Server MBeans are deprecated.
Instead, client classes that interact with WebLogic Server MBeans
should use standard JMX design patterns in which clients use the
<code>javax.management.MBeanServerConnection</code> interface to
discover MBeans, attributes, and attribute types at runtime.
For more  information, see "Developing Manageable Applications with
JMX" on <a href="http://www.oracle.com/technology/products/weblogic/
index.html" shape="rect">http://www.oracle.com/technology/products/
weblogic/index.html</a>.</p>

(com.bea:Name=ThreadPoolRuntime,ServerRuntime=EMGC_ADMINSERVER,Type=
ThreadPoolRuntime)
Enter the index of target/MBean you wish to monitor or press <Ctrl-
C> to quit: 0
Following metric source types are available for selected target(s):
        0: JMX Attributes
        1: JMX Operations
Enter the index of your choice or press <Ctrl-C> to quit: 0
Attributes are:
        0: CompletedRequestCount          Return Value:
java.lang.Long
        1: ExecuteThreadIdleCount          Return Value:
java.lang.Integer
        2: ExecuteThreads        Return Value:
[Lweblogic.management.runtime.ExecuteThread;
        3: ExecuteThreadTotalCount         Return Value:
```

```
java.lang.Integer
        4: HealthState    Return Value: weblogic.health.HealthState
        5: HoggingThreadCount     Return Value: java.lang.Integer
        6: MinThreadsConstraintsCompleted       Return Value:
java.lang.Long
        7: MinThreadsConstraintsPending  Return Value:
java.lang.Integer
        8: Name  Return Value: java.lang.String
        9: Parent        Return Value: javax.management.ObjectName
        10: PendingUserRequestCount      Return Value:
java.lang.Integer
        11: QueueLength  Return Value: java.lang.Integer
        12: SharedCapacityForWorkManagers       Return Value:
java.lang.Integer
        13: StandbyThreadCount   Return Value: java.lang.Integer
        14: Suspended    Return Value: java.lang.Boolean
        15: Throughput   Return Value: java.lang.Double
        16: Type         Return Value: java.lang.String
Select one or more items as comma separated indices: 5,13
Number of possible columns in the resultant metric are 2.
Enter the name for this metric column at index=0 :
[HoggingThreadCount]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [HoggingThreadCount]
Enter the NLSID for column: [HoggingThreadCount]
Enter the UNIT for column "HoggingThreadCount": [millisec, kb
etc.. ]
Do you want to create a threshold for this column <y/n>? [n]
Enter the name for this metric column at index=1 :
[StandbyThreadCount]
Is this column a KEY Column <y/n>? [n]
Is this column for SUMMARY_UI <y/n>? [n]
Enter the label for column: [StandbyThreadCount]
Enter the NLSID for column: [StandbyThreadCount]
Enter the UNIT for column "StandbyThreadCount": [millisec, kb
etc.. ]
Do you want to create a threshold for this column <y/n>? [n]
Do you want periodic collection for this metric <y/n>? [n] y
Enter the collection interval in seconds: 300
Periodic collection interval is: 300 seconds.
Written the metadata xml file: ./metadata/ME#24#myAppME_1.xml.
Creating new file: ./collection/ME#24#myAppME_1.xml.
Updated the default collection file for j2ee_application at
location ./collection/ME#24#myA
ppME_1.xml.
createMextArchive: Adding metadata
createMextArchive: Adding collection file
createMextArchive: Adding mea.xml file

Creating Metric Extension zip archive: ./MEA_ME$myAppME_1.zip
Please import this into Enterprise Manager Cloud Control using the
console.
Exiting...
```

The previous session creates a ZIP file MEA_ME$myAppME_1.zip in the directory specified by the -w option when `jmxcli` is invoked (or in current directory if -w is not specified).

Import this into the Enterprise Manager console as shown below. From the **Enterprise** menu, choose **Monitoring** and then **Metric Extensions** to access the Metric Extensions home page.



After the Management Extension Archive is imported as shown in the preceding example, it can be edited (and modified), tested, published and deployed.

# Surfacing Metrics from a Standalone JVM or Oracle Coherence

Users can also use the mechanism outlined in previous section to create additional metrics that are not available out-of- box for Oracle Coherence or JVM targets and the data for which are available via JMX Mbean attributes.

## Using the Enterprise Manager Console

The procedure is similar to the ones followed in previous section for extending metrics on j2ee_application target types except that you must select target type "JVM" or "Oracle Coherence xxx" in Step 3 for defining the Metric Extension on JVM or Oracle Coherence target types.

## Using JMXCLI

The steps are similar to those for using JMXCLI to define a Metric Extension Archive for custom J2EE applications except that the start-up arguments when `jmxcli` is invoked as follows:

```
emctl jmxcli -t JVM -MEXT  -h adc2180736 -p 6789 -m "*:*"  -w /scratch/TEMP/
```

You must specify target type on which the Metric Extension is defined to be JVM or oracle_coherence as appropriate (instead of the default j2ee_application).

# Monitoring Using RESTful Services

Monitoring REST-compliant Web resources is achieved using the REST fetchlet. For more information about the REST fetchlet, see REST Fetchlet .

# 19

# Using Receivelets

This chapter contains the following sections:

- About Receivelets
- SNMP Receivelet

## About Receivelets

A receivelet is a library that allows Enterprise Manager to receive external notifications sent by third-party network elements. These are notifications that are asynchronously sent and without any requests from the Management Agent.

Usually, the Management Agent data retrieval mechanism is based on a polling model, that is, modular libraries, called fetchlets. Fetchlets collect values of various metrics from their managed targets on a regular basis. The Management Agent then compares the gathered data with user-defined thresholds and generates events when the thresholds were met.

Receivelets are a more efficient way of dealing with these metrics. It depends on the ability of the managed target to detect the condition for its own events, and then communicate with Enterprise Manager only when an event occurs. When this communication happens, the Management Agent uses receivelets to receive the information.

You can use receivelets as a quicker way to get alerts on data that will be eventually collected via fetchlets. You can also use receivelets as a way to send both alerts and data, or just alerts for cases where there is no real data chart associated with the alert.

A receivelet is not a substitute for a fetchlet, but it is another way of collecting data. It is more for immediacy of notification compared to periodic polling that the fetchlet offers. Therefore, if you can fetch data, then use fetchlets to get that data. However, if your server is capable of sending you events or data at a cost lower than that associated with fetchlets, then use receivelets.

A receivelet may be tightly coupled to a particular type of managed target, or may be useful to a broad range of potential targets.

The SNMP receivelet is offered with Enterprise Manager as described in the following section.

## SNMP Receivelet

An SNMP receivelet allows you to receive SNMP trap notifications from third-party network elements, and translate them into a form compatible with Oracle Management Service (OMS).

While monitoring third-party entities in your managed environment, if the third-party entity wants to send a notification to Enterprise Manager, then the SNMP agent of that third-party entity sends a notification to the Management Agent. These notifications

are in the form of SNMP traps that get triggered asynchronously and without any requests from the Management Agent.

Since these traps are based on SNMP, the Management Agent uses SNMP receivelets to receive and translate these SNMP traps into a form compatible with OMS.

When the SNMP traps are received, the SNMP receivelet extracts information pertaining to those object identifiers (OIDs) that are defined in the `<PushDescriptor>` section of the target type metadata file only. For more information about the target type metadata file, see Creating the Target Type Metadata File.

When a customer wants to manage a network element using the SNMP receivelet, they must configure the element's SNMP agent to send traps to the responsible Management Agent's SNMP receivelet. When the SNMP receivelet receives such traps, it translates them to an Enterprise Manager format (such as an event or datapoint based on the *push descriptor* information), and stores that information (in XML files) in the upload directory. The Upload Manager checks for such new files in the upload directory, and then uploads those files to OMS. Then Enterprise Manager accesses OMS to extract the collected information and displays it to the user.

**Receiving SNMP Traps**

To receive SNMP traps, you have to make some configuration settings at the Management Agent side and at SNMP target agent side.

This enables the SNMP targets to send SNMP traps to the SNMP receivelet. When the SNMP traps are received, the SNMP receivelet uses the Push Descriptor properties, such as MatchAgentAddr, MatchEnterprise, and so on, to identify the target and metric for which the traps belongs. Then the SNMP receivelet uses the Push Descriptor properties, such as Event*metric-column* or Event*metric-column*OID, SeverityCode, and so on, to convert the traps into an event. When this happens, the SNMP receivelet uploads the converted event to the Management Repository and it is now available in the Cloud Control console.

**Configuration Settings Required at the Management Agent Side**

By default, the SNMP receivelet listens over UDP on the same port as that of the Management Agent. However, if you want to use a different listening port for the SNMP receivelet, then add the `SnmpRecvletListenNIC(=8002)` entry to the `emd.properties` file.

**Configuration Settings Required at SNMP Target Agent Side**

Configure the SNMP target agent to send its traps to the SNMP target agent's host name and port.

**Input Parameters**

**Table 19-1    SNMP Receivelet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| MatchEnterprise | String | **Note**: For Push Descriptors that intend to match SNMPv1 traps only.<br>OID used to define the trap being sent.<br>**Note**: If MatchEnterprise is present, then you must include MatchGenericTrap and MatchSpecificTrap also. | Required (SNMPv1) |
| MatchGenericTrap | String | Code for a generic SNMP trap. | Required (SNMPv1) |
| MatchSpecificTrap | String | Trap defined in a MIB (not one of the generic traps), the ID assigned in that MIB. | Required (SNMPv1) |
| MatchTrapOID | String | **Note**: For Push Descriptors that intend to match SNMPv2 or SNMPv3 traps.<br>For an SNMPv2 or SNMPv3 trap, this is the OID assigned to the NOTIFICATION-TYPE in the MIB definition of the trap. | Required (SNMPv2, SNMPv3) |
| MatchAgentAddr | String | IP address of the generating SNMP agent, as sent in the trap. | Required |
| MatchVarBind | String | If this parameter is present, its value is an OID. If a trap is received that matches the other Match* parameters, it still must have this OID on its received varbind list. Otherwise, it will not generate an Enterprise Manager event or datapoint. | Optional |
| UseCredential | String | Specifies credential use. If it is set to TRUE, then the receivelet will accept SNMPv3 traps using a set of target SNMPv3Creds sent to the receivelet as a CredentialRef. Also, if UseCredential is set, then the string property "hostname" and the numeric property "PORT" must be specified; their values will be used for engine-id discovery. | Optional |
| Event*metric-column* | String | Specifies that, on receiving this trap, the receivelet must generate a severity on this metric column. The value of *metric-column* must be the value of this parameter. (This case is useful where the expected values of the Enterprise Manager metric are not the same as the triggering SNMP variable.) | Required, if events have to be generated. However, if Event*metric-column*OID is provided, then this is not required. |
| Event*metric-column*OID | String | Specifies that, on receiving this trap, the receivelet must generate a severity on this metric column. The value of the metric column should be taken from the varbind in the trap with an OID equal to the value of this parameter. | Required, if events have to be generated. However, if Event*metric-column* is provided, then this is not required. |

**Table 19-1    (Cont.) SNMP Receivelet Input Parameters**

| Parameter | Type | Description | Use |
| --- | --- | --- | --- |
| SeverityCode | String | Specifies the level at which the severity must be generated. The value of this parameter must be 'CRITICAL', 'WARNING', or 'CLEAR'.<br><br>**Note**: SeverityCode or SeverityCodeOID must be present if events are to be generated when the trap is received. However, if the Push Descriptor intends to generate a metric datapoint (and specifies one or more Data*metric-column*OID properties), then it must *not* have a SeverityCode or SeverityCodeOID property. | Required. However, if SeverityCodeOID is provided, or the Push Descriptor is generating a metric datapoint, then this is not required. |
| SeverityCodeOID | String | Specifies the level at which the severity should be generated. If the varbind in the trap with OID equal to the value of this parameter is one of the strings 'CRITICAL', 'WARNING', or 'CLEAR', then the severity must be generated at that level; otherwise, no severity must be generated. (This parameter is only used if you are designing a trap exclusively for use with Enterprise Manager, but can be useful in this case.)<br><br>**Note**: SeverityCode or SeverityCodeOID must be present if events are to be generated when the trap is received. However, if the Push Descriptor intends to generate a metric datapoint (and specifies one or more Data*metric-column*OID properties), then it must *not* have a SeverityCode or SeverityCodeOID property. | Required. However, if SeverityCode is provided, or the PushDescriptor is generating a metric datapoint, then this is not required. |
| Data*metric-column*OID | String | Specifies that, on receiving this trap, the receivelet must generate a datapoint on the metric, for which the value of this metric column should be taken from the varbind in the trap with OID equal to the value of this parameter. (An SNMP Push Descriptor can have many Data* parameters, in which case a single row will be returned, with all specified columns populated from the appropriate varbind in the trap. An SNMP Push Descriptor *cannot* have both a Data* parameter and a Severity* parameter, nor can it have multiple Severity* parameters.) | Required, if datapoints have to be generated. |
| Key*metric-column*OID | String | Severity or datapoint generated by this Push Descriptor should contain a key-value for this metric column. The key-value should be taken from the varbind in the trap with OID equal to the value of this parameter. For every key-column in the metric definition, there must be a Key* parameter in the Push Descriptor. | Optional |

**Table 19-1    (Cont.) SNMP Receivelet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| Context*metric-column*OID | String | If the Push Descriptor generates a severity, then the severity must contain a value for this metric column in its event context. The value should be taken from the varbind in the trap with OID equal to the value of this parameter. If the Push Descriptor generates a datapoint, then this parameter is ignored. | Optional. This can be used for severities only. |

### Example

The following example shows what a trap from a vendor-specific router looks like.

**Example: Trap from a Vendor-Specific Router**

```
ascendLinkDown    TRAP-TYPE
   ENTERPRISE     ascend
   VARIABLES      { ifIndex, ifAdminStatus, ifOperStatus, ifType,
                     ifName }
   DESCRIPTION     "This trap is in addition to the generic linkDown
             trap defined in RFC1215.  This trap provides
             additional information such as ifAdminStatus,
             ifOerStatus, ifName, slotIfSlotIndex, slotIfItemIndex.
             This is an Alarm class trap and it can
             be enabled/disabled via alarmEnabled and/or
             ascendLinkDownTrapEnabled in trap profile.
             This trap is sent only if rfc1215 linkDown trap is
generated."
   ::= 50
```

The following example shows how the trap will be received by the Management Agent. Note that <x> in this example is the value of *ifIndex* that identifies the particular interface that's having problems.

**Example: Trap Received by the Management Agent**

```
Message:
    version: 0
    community: 'public'
    Trap-PDU:
        enterprise: enterprises.ascend (1.3.6.1.4.1.529)
        agent-addr: 138.2.204.10
        generic-trap: 6
        specific-trap: 50
        time-stamp: <timestamp from router's sysUptime>
        variable-bindings:
            Name: ifIndex.<x> (1.3.6.1.2.1.2.2.1.1.<x>)
            Type: INTEGER
            Value: <x>

            Name: ifAdminStatus.<x> (1.3.6.1.2.1.2.2.1.7.<x>)
            Type: INTEGER
```

```
Value: up (1)

Name: ifOperStatus.<x> (1.3.6.1.2.1.2.2.1.8.<x>)
Type: INTEGER
Value: down (2)

Name: ifType.<x> (1.3.6.1.2.1.2.2.1.3.<x>)
Type: INTEGER
Value: iso88023-csmacd (7)

Name: ifName.<x> (1.3.6.1.2.1.2.2.1.31.<x>)
Type: DisplayString
Value: 'eth0'
```

The following example shows how the metric can be defined in the target type metadata file.

**Example: Metric Defined in the Target Type Metadata File**

```
<Metric NAME="interfaces" TYPE="TABLE">
    <TableDescriptor>
        <ColumnDescriptor NAME="name" TYPE="STRING" IS_KEY="TRUE"/>
        <ColumnDescriptor NAME="type" TYPE="NUMBER" IS_KEY="FALSE"/>
        <ColumnDescriptor NAME="status" TYPE="NUMBER" IS_KEY="FALSE"/>
        <ColumnDescriptor NAME="configured_status" TYPE="NUMBER"
IS_KEY="FALSE"/>
    </TableDescriptor>
</Metric>
```

**To Receive SNMPV1 Trap**

The following example shows how the push descriptor can be defined in the target type metadata file to trigger a severity.

**Example: Push Descriptor in the Target Type Metadata File For Triggering a Severity**

```
<PushDescriptor RECVLET_ID="SNMPTrap">
   <Property NAME="MatchEnterprise" SCOPE="GLOBAL">1.3.6.1.4.1.529</
Property>
   <Property NAME="MatchGenericTrap" SCOPE="GLOBAL">6</Property>
   <Property NAME="MatchSpecificTrap" SCOPE="GLOBAL">50</Property>
   <Property NAME="MatchAgentAddr" SCOPE="INSTANCE">AdminAddress</
Property>
   <Property NAME="SeverityStatusOID"
SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.8</Property>
   <Property NAME="KeyNameOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.31</
Property>
   <Property NAME="ContextTypeOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.3</
Property>
   <Property NAME="ContextConfigured_statusOID"
SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.7</Property>
   <Property NAME="SeverityCode" SCOPE="GLOBAL">CRITICAL</Property>
```

```
        <CredentialRef NAME="monCreds">monCredentials</CredentialRef>
</PushDescriptor>
```

The following example shows how the push descriptor can be defined in the target type metadata file to trigger a datapoint, which would specify the reporting of data on the same trap, with *ifName* as the key-column and the other three as data columns.

**Example: Push Descriptor in the Target Type Metadata File For Triggering a Datapoint**

```
<PushDescriptor RECVLET_ID="SNMPTrap">
    <Property NAME="MatchEnterprise" SCOPE="GLOBAL">1.3.6.1.4.1.529</
Property>
    <Property NAME="MatchGenericTrap" SCOPE="GLOBAL">6</Property>
    <Property NAME="MatchSpecificTrap" SCOPE="GLOBAL">50</Property>
    <Property NAME="MatchAgentAddr" SCOPE="INSTANCE">AdminAddress</
Property>
    <Property NAME="KeyNameOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.31</
Property>
    <Property NAME="DataStatusOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.8</
Property>
    <Property NAME="DataTypeOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.3</
Property>
    <Property NAME="DataConfigured_statusOID"
SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.7</Property>
<CredentialRef NAME="monCreds">monCredentials</CredentialRef>
</PushDescriptor>
```

**To Receive SNMPV2 Notifications**

The following example shows how the push descriptor can be defined in the target type metadata file to trigger a severity:

**Example: Push Descriptor in the Target Type Metadata File For Triggering a Severity**

```
<PushDescriptor RECVLET_ID="SNMPTrap">
    <Property NAME="MatchTrapOID" SCOPE="GLOBAL">1.3.6.1.4.1.529.50</
Property>
    <Property NAME="MatchAgentAddr" SCOPE="INSTANCE">AdminAddress</
Property>
    <Property NAME="SeverityStatusOID"
    SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.8</Property>
    <Property NAME="KeyNameOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.31</
Property>
    <Property NAME="ContextTypeOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.3</
Property>
    <Property NAME="ContextConfigured_statusOID"
    SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.7</Property>
    <Property NAME="SeverityCode" SCOPE="GLOBAL">CRITICAL</Property>
    <CredentialRef NAME="monCreds">monCredentials</CredentialRef>
</PushDescriptor>
```

The following example shows how the push descriptor can be defined in the target type metadata file to trigger a datapoint, which would specify the reporting of data on the same trap, with *ifName* as the key-column and the other three as data columns.

**Example: Push Descriptor in the Target Type Metadata File For Triggering a Datapoint**

```
<PushDescriptor RECVLET_ID="SNMPTrap">
    <Property NAME="MatchEnterprise" SCOPE="GLOBAL">1.3.6.1.4.1.529</
Property>
    <Property NAME="MatchGenericTrap" SCOPE="GLOBAL">6</Property>
    <Property NAME="MatchSpecificTrap" SCOPE="GLOBAL">50</Property>
    <Property NAME="MatchAgentAddr" SCOPE="INSTANCE">AdminAddress</
Property>
    <Property NAME="KeyNameOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.31</
Property>
    <Property NAME="DataStatusOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.8</
Property>
    <Property NAME="DataTypeOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.3</
Property>
    <Property NAME="DataConfigured_statusOID"
    SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.7</Property>
    <CredentialRef NAME="monCreds">monCredentials</CredentialRef>
</PushDescriptor>
```

**To Receive SNMPV3 Notifications**

The following example shows how the push descriptor can be defined in the target type metadata file to trigger a severity.

**Example: Push Descriptor in the Target Type Metadata File For Triggering a Severity**

```
<PushDescriptor RECVLET_ID="SNMPTrap">
    <Property NAME="MatchTrapOID" SCOPE="GLOBAL">1.3.6.1.4.1.529.50</
Property>
    <Property NAME="MatchAgentAddr" SCOPE="INSTANCE">AdminAddress</
Property>
    <Property NAME="SeverityStatusOID"
    SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.8</Property>
    <Property NAME="KeyNameOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.31</
Property>
    <Property NAME="ContextTypeOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.3</
Property>
    <Property NAME="ContextConfigured_statusOID"
    SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.7</Property>
    <Property NAME="SeverityCode" SCOPE="GLOBAL">CRITICAL</Property>
    <Property NAME="hostname" SCOPE="INSTANCE">snmpHost</Property>
    <Property NAME="PORT" SCOPE="INSTANCE">snmpPort</Property>
    <CredentialRef NAME="monCreds">monCredentials</CredentialRef>
</PushDescriptor>
```

The following example shows how the push descriptor can be defined in the target type metadata file to trigger a datapoint, which would specify the reporting of data on the same trap, with *ifName* as the key-column and the other three as data columns.

**Example: Push Descriptor in the Target Type Metadata File For Triggering a Datapoint**

```
<PushDescriptor RECVLET_ID="SNMPTrap">
    <Property NAME="MatchEnterprise" SCOPE="GLOBAL">1.3.6.1.4.1.529</
Property>
    <Property NAME="MatchGenericTrap" SCOPE="GLOBAL">6</Property>
    <Property NAME="MatchSpecificTrap" SCOPE="GLOBAL">50</Property>
    <Property NAME="MatchAgentAddr" SCOPE="INSTANCE">AdminAddress</
Property>
    <Property NAME="KeyNameOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.31</
Property>
    <Property NAME="DataStatusOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.8</
Property>
    <Property NAME="DataTypeOID" SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.3</
Property>
    <Property NAME="DataConfigured_statusOID"
    SCOPE="GLOBAL">1.3.6.1.2.1.2.2.1.7</Property>
    <Property NAME="hostname" SCOPE="INSTANCE">snmpHost</Property>
    <Property NAME="PORT" SCOPE="INSTANCE">snmpPort</Property>
    <CredentialRef NAME="monCreds">monCredentials</CredentialRef>
</PushDescriptor>
```

The following example shows how the monCredentials is defined in target type metadata file

**Example: monCredentials in the Target Type Metadata File**

```
<CredentialInfo>
    <CredentialSet NAME="monCredentials" USAGE="MONITORING">
        <AllowedCredType TYPE="SNMPV1Creds" />
        <AllowedCredType TYPE="SNMPV3Creds" />
    </CredentialSet>
</CredentialInfo>
```

SNMPV1Creds or SNMPV3Creds values for the respective targets must be set from the Cloud Control console by selecting **Setup**, then **Security**, and then **Monitoring Credentials**.

To receive an SNMPV1 trap or SNMPV2 Notification, the user must choose SNMPV1Creds. Choosing SNMPV1Creds will ask for "Community String" parameter value. Appropriate community string values need to be set by the user.

To receive an SNMPV3 Notification, the user must choose SNMPV3Creds. Choosing SNMPV3Creds will ask for "UserName", "Auth Password", "Auth Protocol" and "Privacy Password" parameter values. The user must set the required values according to the secLevel they want to use.

**Example: SNMV1 Trap Received by the Management Agent**

```
Message:
version: 0
community: 'public'
Trap-PDU:
enterprise: enterprises.ascend (1.3.6.1.4.1.529)
```

```
agent-addr: 138.2.204.10
generic-trap: 6
specific-trap: 50
time-stamp: <timestamp from router's sysUptime>
variable-bindings:
Name: ifIndex.<x> (1.3.6.1.2.1.2.2.1.1.<x>)
Type: INTEGER
Value: <x>
Name: ifAdminStatus.<x> (1.3.6.1.2.1.2.2.1.7.<x>)
Type: INTEGER
Value: up (1)
Name: ifOperStatus.<x> (1.3.6.1.2.1.2.2.1.8.<x>)
Type: INTEGER
Value: down (2)
Name: ifType.<x> (1.3.6.1.2.1.2.2.1.3.<x>)
Type: INTEGER
Value: iso88023-csmacd (7)
Name: ifName.<x> (1.3.6.1.2.1.2.2.1.31.<x>)
Type: DisplayString
Value: 'eth0'
```

**Example: SNMV2/SNMPV3 Notification Received by the Management Agent**

```
Message: Recevied from address 138.2.204.10 (ip address from UDP layer)
version: 1 (or 3 i.e 1 for snmpv2 and 3 for snmpv3 notification)
Security params : (community if SNMPV2, SNMPV3 security params if
SNMPV3)
Trap-PDU:
variable-bindings:
Name: sysUpTime.0 (.1.3.6.1.2.1.1.3.0)
Type: INTEGER
Value: 43053404
Name: snmpTrapOID.0 (1.3.6.1.6.3.1.1.4.1.0)
Type: Object Identifier
Value: 1.3.6.1.4.1.529.50
Name: ifAdminStatus.<x> (1.3.6.1.2.1.2.2.1.7.<x>)
Type: INTEGER
Value: up (1)
Name: ifOperStatus.<x> (1.3.6.1.2.1.2.2.1.8.<x>)
Type: INTEGER
Value: down (2)
Name: ifType.<x> (1.3.6.1.2.1.2.2.1.3.<x>)
Type: INTEGER
Value: iso88023-csmacd (7)
Name: ifName.<x> (1.3.6.1.2.1.2.2.1.31.<x>)
Type: DisplayString
Value: 'eth0'
```

**Notes**

- The target type metadata file can have multiple metrics with push descriptor definitions. Also, a single metric can have multiple push descriptors attached.

- For event format, ensure that your push descriptor defines only one Event* parameter indicating one metric column as described in Table 19-1. The event

push descriptor can have one or more Context* parameters to indicate additional metric column values to send as part of an AlertContext.

- For datapoint format, ensure that your push descriptor defines one or more Data* properties as described in Table 19-1 and demonstrated in the Push Descriptor in the Target Type Metadata File For Triggering a Datapoint example and the Push Descriptor in the Target Type Metadata File For Triggering a Datapoint example in this chapter.

# 20

# Using Fetchlets

This chapter contains the following sections:

## About Fetchlets

Enterprise Manager data retrieval is handled through predefined "fetchlets." A fetchlet is a parametrized data access mechanism that takes arguments (for example, a script, a SQL statement, a target instance's properties) as input and returns formatted data. Each fetchlet handles a specific type of data access. The fetchlets supplied with Enterprise Manager provide data retrieval capability for the most common data access methods, such as SQL, SNMP (Simple Network Management Protocol), HTTP, and DMS (Dynamic Monitoring Service). To handle more complex data access requirements, Enterprise Manager also provides an OS command fetchlet that allows developers to implement custom metric collection methods.

The following sections describe the fetchlets supplied with Enterprise Manager:

## OS Command Fetchlets

The operating system (OS) command fetchlets allow you to obtain metric data by executing OS commands (either individually or from scripts) that return a standard out (stdout) data stream.

Three OS command fetchlets are available:

- OS Fetchlet (raw)

- OSLines Fetchlet (split into lines)
- OSLineToken Fetchlet (tokenized lines)

# OS Fetchlet

The OS fetchlet executes a given OS command and returns the command's output in a single cell table.

**Input Parameters**

**Table 20-1    OS Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| command | string | Operating system command to be executed. | Required |
| ENV*name* | string | OSFetchlet parameters starting with "ENV" appear in the execution environment for the command as *name* environment variables | Zero or more |
| errStartsWith | String | When defined, this property allows you to define a custom prefix for error messages. If this property is not defined, the OSFetchlet defaults to "em_error=" as the message prefix. | Optional |
| script | string | Specifies the script to be executed if *command* property only provides an interpreter. For example, *command* might consist of "perl." *script* is then used to specify the particular perl script to be run.<br><br>Although scripts can be specified directly from the *command* parameter, using the *script* parameter adds to stylistic clarity and readability when defining a target type metadata file. | Optional |
| args | string | A property that is taken as one or more arguments to the command and script properties.<br><br>Although args can be specified directly from the command parameter, using the args parameter adds to stylistic clarity and readability when defining a target type metadata file. | Optional |
| separateErrorStream | boolean | If an error occurs while executing a *command*, this property instructs the fetchlet whether to return both the stdout and stderr to the user as an error message. When set to TRUE, only stderr output is sent to the user as an error message when there is a *command* error. When set to FALSE (default value), both the stdout and the stderr are sent to the user as an error message upon *command* failure. | Optional. (TRUE/FALSE) |

**Table 20-1    (Cont.) OS Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| em_metric_timeout | integer | Metric timeout period (in seconds). After the timeout period has finished, the Management Agent returns a timeout exception and terminates any child processes that may have been created. The Management Agent *does not* terminate any of the grandchild processes.<br><br>Specify "-1" for no timeout period. | Optional |

**Example**

You want to obtain metric data by executing the UNIX *echo* command.

To run the command from the shell environment, enter:

```
echo Line 1|some more|even more\nLine 2\n\nLine 4|a little more|\n|Line 5\n|Line
6|\n|Line 7|again|\nLine 8|the|end
```

The *echo* command produces the following standard output:

```
Line 1|some more|even more
Line 2

Line 4|a little more|
|Line 5
|Line 6|
|Line 7|again|
Line 8|the|end
```

Using the OS fetchlet with the given example command.

The fetchlet returns the following 1 x 1 table:

**Figure 20-1    Table Returned by the OS Fetchlet**



The raw output of the OS command is returned. Any standard error output is appended to the standard output.

**Error Handling**

Any problems encountered launching the *command* (For example, the *command* program no longer exists) results in an oracle.sysman.emSDK.emd.fetchlet.MetricSourceException wrapping a java.io.IOException. If the command exits with a non-zero exit value, the fetchlet throws an oracle.sysman.emSDK.emd.fetchlet.MetricSourceException wrapping an oracle.sysmand.emd.fetchlets.CommandFailedException.

**Notes**

Commands are *not* executed as if they are being run in a shell. This means that common shell symbols do not work, including piping, output redirection, and backgrounding.

Commands cannot read from standard input.

The fetchlet blocks and waits for the command to finish.

# OSLines Fetchlet (split into lines)

The OS Lines fetchlet executes a given OS command and tokenizes the OS command's output. The output is tokenized by lines. The fetchlet returns the tokens in a single column table. The *n*th row in the table represents the *n*th line in the output of the OS command.

To get the raw, untokenized output of an OS command, use the OS fetchlet. To get the output of an OS command tokenized by lines and each line tokenized by a given delimiter, see the OS Line Token fetchlet.

**Input Parameters**

**Table 20-2    OSLines Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| command | string | Operating system command to be executed. | Required |
| startsWith | string | Only lines starting with this string are included in the result. | Optional; default = "" (all lines are included) |
| ENV*name* | string | Parameters starting with "ENV" appear in the execution environment for the command as *name* environment variables | Zero or more of these |
| errStartsWith | string | When defined, this property allows you to define a custom prefix for error messages. If this property is not defined, the OSFetchlet defaults to "em_error=" as the message prefix. | Optional |
| script | string | Specifies the script to be executed if *command* property only provides an interpreter. For example, *command* might consist of "perl." *script* is then used to specify the particular perl script to be run.<br><br>Although scripts can be specified directly from the *command* parameter, using the *script* parameter adds to stylistic clarity and readability when defining a target type metadata file. | Optional |

**Table 20-2 (Cont.) OSLines Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| args | string | A property that is taken as one or more arguments to the *command* and *script* properties.<br><br>Although args can be specified directly from the *command* parameter, using the *args* parameter adds to stylistic clarity and readability when defining a target type metadata file. | Optional |
| separateErrorStream | boolean | If an error occurs while executing a command, this property instructs the fetchlet to whether to return both the stdout and stderr to the user as an error message. When set to TRUE, only stderr output is sent to the user as an error message when there is a command error. When set to FALSE (default value), both the stdout and the stderr are sent to the user as an error message upon command failure. | Optional. (TRUE/FALSE) |
| em_metric_timeout | integer | Metric timeout period (in seconds). After the timeout period has finished, the Management Agent returns a timeout exception and terminates any child processes that may have been created. The Management Agent DOES NOT kill any of the grandchild processes.<br><br>Specify "-1" for no timeout period. | Optional |

**Example**

Take the following UNIX command:

```
echo Line 1|some more|even more\nLine 2\n\nLine 4|a little more|\n|Line 5\n|Line
6|\n|Line 7|again|\nLine 8|the|end
```

It produces the following output:

```
Line 1|some more|even more
Line 2

Line 4|a little more|
|Line 5
|Line 6|
|Line 7|again|
Line 8|the|end
```

Running OSLinesFetchlet with the given example command produces the following single column table.

**Figure 20-2    Table Returned by the OS LINES Fetchlet**



Note that without content, "\n" results in a blank line inserted between Line 2 and Line 4.

> **Note:**
>
> Commands are *not* executed as if they are being run in a shell. This means that common shell symbols do not work, including piping, output redirection, and backgrounding.
>
> Commands cannot read from standard input.
>
> The fetchlet blocks and waits for the command to finish.
>
> The standard output of the command is captured and the standard error is captured and appended to the standard output.
>
> Lines are tokenized using "\n".

## OSLineToken Fetchlet (tokenized lines)

The OS Line Token fetchlet executes a given OS command and tokenizes the output of the OS command. The output is tokenized first by lines, and then each line is tokenized by a given delimiter set. The fetchlet returns the tokens in a table. The $n$th row in the table represents the $n$th line in the output of the OS command. The $n$th column in the table represents the $n$th token in a line as determined by the given delimiter set.

To get the raw, untokenized output of an OS command, see OS Fetchlet.

**Input Parameters**

**Table 20-3    OSLineToken Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| command | String | Operating system command to be executed. | Required |

**Table 20-3    (Cont.) OSLineToken Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| delimiter | String | Set of characters that act as delimiters to tokenize the lines | Optional; default = "" (just breaks output into lines) |
| startsWith | String | Only lines starting with this string are included in the result | Optional; default = "" (all lines are included) |
| ENV*name* | String | Parameters starting with "ENV" appear in the execution environment for the command as *name* environment variables | Zero or more of these |
| errStartsWith | String | When defined, this property allows you to define a custom prefix for error messages. If this property is not defined, the OSFetchlet defaults to "em_error=" as the message prefix. | Optional |
| script | String | Specifies the script to be executed if command property only provides an interpreter. For example, *command* might consist of "perl." The script is then used to specify the particular perl script to be run.<br><br>Although scripts can be specified directly from the command parameter, using the script parameter adds to stylistic clarity and readability when defining a target type metadata file. | Optional |
| args | String | A property that is taken as one or more arguments to the *command* and *script* properties.<br><br>Although args can be specified directly from the *command* parameter, using the args parameter adds to stylistic clarity and readability when defining a target type metadata file. | Optional |
| separateErrorStream | Boolean | If an error occurs while executing a *command*, this property instructs the fetchlet to whether to return both the stdout and stderr to the user as an error message. When set to TRUE, only stderr output is sent to the user as an error message when there is a *command* error. When set to FALSE (default value), both the stdout and the stderr are sent to the user as an error message upon *command* failure. | Optional. (TRUE/FALSE) |

**Table 20-3    (Cont.) OSLineToken Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| em_metric_timeout | Integer | Metric timeout period (in seconds). After the timeout period has finished, the Management Agent returns a timeout exception and terminates any child processes that may have been created. The Management Agent DOES NOT kill any of the grandchild processes.<br><br>Specify "-1" for no timeout period. | Optional |

**Example**

Take the following UNIX command:

```
echo Line 1|some more|even more\nLine 2\n\nLine 4|a little more|\n|Line 5\n|Line
6|\n|Line 7|again|\nLine 8|the|end
```

It produces the following output:

```
Line 1|some more|even more
Line 2

Line 4|a little more|
|Line 5
|Line 6|
|Line 7|again|
Line 8|the|end
```

Running OSLineTokenFetchlet with the given example command and a single character "|" for the delimiter generates the following table:

**Figure 20-3    Table Returned by the OS Token Lines Fetchlet**

| Line 1 | some more | even more |
|---|---|---|
| Line 2 | | |
| | | |
| Line 4 | a little more | |
| Line 5 | | |
| Line 6 | | |
| Line 7 | again | |
| Line 8 | the | end |

**Error Handling**

Any problem launching the command (unable to find the command program) results in an `oracle.sysman.emSDK.emd.fetchlet.MetricSourceException` wrapping a `java.io.IOException`.

If the command exits with a non-zero exit value, the fetchlet throws a `oracle.sysman.emSDK.emd.fetchlet.MetricSourceException` wrapping a `oracle.sysmand.emd.fetchlets.CommandFailedException`.

**Notes**

Commands are *not* executed as if they are being run in a shell. This means that common shell symbols do not work, including piping, output redirection, and backgrounding.

The fetchlet promptly closes the input stream to the running command.

The fetchlet blocks and waits for the command to finish.

Lines are tokenized using "\n".

The delimiter can be a single character or a set of characters. For example, it can be "|+_", if the line should be broken up by pipes, pluses, and underscores. If two or more delimiters are together in the output text, such as "||" or "+|+", then it is as if there are empty string tokens between them. These empty strings get columns in the result table. It is *not* considered that there are empty strings preceding a delimiter that starts a line or following a delimiter that ends a line.

In order to express non-printable characters in the delimiter set (such as tabs) in XML, use "&#xHH;" where H is the hexadecimal identifier for the character.

## Invoke an OS Fetchlet as a Specific User for Metric Collection

Depending on requirements, your plug-in may need to utilize the OS fetchlet to invoke a pre-existing script on the Management Agent monitoring a target to collect data for a specific metric as a specific user; that is, as a user other than the default "oracle" user.

Enterprise Manager supports the use of Privilege Delegation Providers (sudo and powerbroker) to invoke metric collections as a specific user. Enabling PDP for a plug-in requires credential setup on both the plug-in and on hosts where the target(s) being monitored are deployed.

In your plug-in, you must set the credential reference in the metric definition in the target metadata file. In the example, example Credentialref line has "your_cred". This value refers to monitoring credential set name.

**Example: Credential Reference in Target Metadata**

```
<TargetMetadata TYPE="my_type" NAME="my_target_name">
  ...
  <Metric NAME="my_special_metric" TYPE="TABLE">
    <TableDescriptor>
       <ColumnDescriptor NAME="test" TYPE="STRING"/>
    </TableDescriptor>
    <QueryDescriptor FETCHLET_ID="OS">
      <Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl</Property>
      <Property NAME="script" SCOPE="GLOBAL">%scriptsDir%/your_
       script.pl</Property>
      <CredentialRef NAME="OSCreds">your_cred</CredentialRef>
    </QueryDescriptor>
  </Metric>
  ...
</TargetMetadata>
```

On the Management Agent monitoring the target, a referenced credential type must be created that points to host:HostCreds, and allow the monitoring credential set be of the new type that you add. See *Oracle Enterprise Manager Cloud Control Administrator's Guide* for details on privilege delegation setup using Enterprise Manager Cloud Control.

The credential data will be persisted to the target metadata file (target.xml) for the Management Agent monitoring the target.

The following example defines the referenced credential type MyHostCreds in target.xml, which is of the credential type host:HostCreds.

### Example: Credential Type Definition in Management Agent

```
<Target TYPE="<removed>" NAME="<removed>" DISPLAY_NAME="<removed>"
ON_HOST="" EMD_URL="https://<removed>/emd/main/" TIMEZONE_REGION=""
IDENTIFIER="TARGET_GUID=<removed>">
...
<CredentialType NAME="MyHostCreds"><CredentialTypeRef
REF_NAME="HostRef" REF_TYPE="HostCreds" REF_TARGETTYPE="host"
ASSOCIATION="host"><CredentialTypeRefColumn NAME="HostUserName"
REF_TYPECOLUMN="HostUserName"/><CredentialTypeRefColumn NAME="HostPassword"
REF_TYPECOLUMN="HostPassword"/></CredentialTypeRef></CredentialType>
...
</Target>
```

When monitoring credentials are updated Cloud Control (via Setup->Security->Monitoring Credentials), the data shown above will be updated on the Management Agent automatically.

The next example defines the HostMonCredSet monitoring credential set, which is of type MyHostCreds (and therefore type host:HostCreds)

```
<CredentialSet NAME="HostMonCredSet" CREDENTIAL_TYPE="MyHostCreds"
USAGE="MONITORING"><AllowedCredType TYPE="MyHostCreds"/>
</CredentialSet>
```

# SQL Fetchlet

The SQL fetchlet executes a given SQL statement on a given database as a given user and returns the table result.

**Input Parameters**

**Table 20-4    SQL Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|-----------|------|-------------|-----|
| Connection Information | | | |
| MachineName | String | Database host | Required |
| Port | Integer | Database port | Required |
| SID | String | Database SID | Required unless ServiceName is specified |
| ServiceName | String | Database ServiceName | Required unless SID is specified |

**Table 20-4    (Cont.) SQL Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| OracleHome | String | Database's Oracle Home (used in conjunction with OidRepSchemaName). | Required when OidRepSchema Name is used. |
| Credential Information | | | |
| UserName | String | user name | Required |
| password | String | user password | Optional; default is "" |
| Role | String | Role used when connecting to the database (e.g., SYSDBA) | Optional; allowed choices are SYSDBA, SYSOPER, and NORMAL (default) |
| General | | | |
| STATEMENT | String | SQL statement or PL/SQL block | Required unless FILENAME is specified. |
| FILENAME | String | Full path of the file containing the SQL statement or PL/SQL block | Required unless STATEMENT is specified. |
| NUMROWS | Integer | Maximum number of rows to output. | Required |
| Bind Parameters | | | |
| SQLINPARAM<position> | String | Value of input parameter at position <position> | Zero or more, one for each input parameter. |
| SQLOUTPARAMPOS | Integer | Position of output parameter, if it exists | There can be exactly one output parameter, if it exists. |
| SQLOUTPARAMTYPE | String | Type of the output parameter, if it exists. | There can be exactly one output parameter type, if it exists. |
| transpose | TRUE/FALSE | If TRUE, the result is transposed: rows to columns and columns to rows. | |

**Notes**

The SQL statement or PL/SQL block can be specified either through the STATEMENT property, or via a file whose name is provided through the FILENAME property.

The SQL fetchlet supports input parameters. Input and output parameters are indicated in the SQL/PLSQL text in the usual way, by using ":<number>". Input parameters can be used to bind values to both SQL queries and PL/SQL blocks.

Input parameter values are specified as properties of the form SQLINPARAM<position>. There can be any number of input parameters. The input parameters need to be scalar: input parameters of type ARRAY and STRUCT are not supported.

The SQL fetchlet supports the execution of anonymous PL/SQL blocks (which may call other functions or procedures) to retrieve data. When executing a block of PL/SQL, data is returned to the fetchlet by means of an OUT parameter. There can be exactly one out parameter. It must be of type SQL_CURSOR (a PL/SQL REF CURSOR), or it must be a named type that represents an array of objects. In the latter case, each field of the object represents one column of the table; and each object instance in the array represents one complete row in the table. The OUT parameter position and type are specified by means of the properties SQLOUTPARAMPOS and SQLOUTPARAMTYPE. If an OUT parameter is specified, then the fetchlet assumes it is executing PL/SQL and treats the STATEMENT property as an anonymous PL/SQL block.

> **Note:**
>
> When using a SQLOUTPARAMTYPE of type 'ARRAY', you must identify the array as follows:
>
> - If you create the array type specified in the SQLOUTPARAMTYPE from SQL*Plus or any utility *without* using double quotation marks to surround the array name, then you must specify the array name using all upper-case letters in the target metadata file for this property. The reason for this because the RDBMS automatically changes the array name to all upper-case.
>
> - If you create the array type specified in the SQLOUTPARAMTYPE from SQL*Plus or any utility using double-quotation marks to surround the array name, then the RDBMS retains the case specified. For this reason, users must specify the array name using the same case used in the target metadata file.

If no OUT parameter is specified, the fetchlet assumes that it is executing a SQL query.

Note that all input parameters to the SQL fetchlet are strings. This means that all other datatypes will have to be converted to strings. This is straightforward for datatypes such as numbers, but not, for example, dates and timestamps. You can pass an absolute date or timestamp by passing a character representation of the value (using a DateFormat class). There is no way currently to pass in a date function, such as SYSDATE or SYSDATE+1. In such case, you could embed the date argument directly in the SQL, for example:

```
begin func1(:1, :2, SYSDATE); end;
```

The other caveat is passing null arguments to a procedure. Consider the following SQL:

```
STATEMENT=begin func1(:1,:2); end;
SQLINPARAM1=null
SQLOUTPARAMPOS=2
SQLOUTPARAMTYPE=fooret
```

Assume that the first argument is intended to be a varchar2. By parameterizing it and passing 'null' as the first argument, what we are really doing is passing the \*string\* 'null' to the argument, and not a null value. If you intend to pass a null value, do the following:

```
STATEMENT=begin func1(null, :1); end;
SQLOUTPARAMPOS=1
SQLOUTPARAMTYPE=fooret
```

**Examples**

The following properties execute a query (get all users) with no parameters:

**Example 1: Query With No Parameters**

```
MachineName=skini-pc
Port=1521
SID=o817
UserName=scott
password=tiger
STATEMENT=select * from all_users;
NUMROWS=30
```

The following properties execute a query (get the first few objects of a specified type owned by a specified user) with input parameters:

**Example 2: Query With Input Parameters**

```
MachineName=skini-pc
Port=1521
SID=o817
UserName=scott
password=tiger
STATEMENT=select * from all_objects where owner=:1 and object_type=:2 and
rownum<:3tt>
NUMROWS=30
SQLINPARAM1=SYSTEM
SQLINPARAM2=INDEX
SQLINPARAM3=10
```

The following example executes a PL/SQL procedure that returns a cursor and has input parameters:

**Example 3: PL/SQL Procedure With Input Parameters**

```
achineName=skini-pc
Port=1521
SID=o817
UserName=scott
password=tiger
STATEMENT=begin :1 := skini_junk.func1(:2); end;
NUMROWS=30
SQLINPARAM2=SYSTEM
SQLOUTPARAMPOS=1
SQLOUTPARAMTYPE=sql_cursor
```

The following example specifies a PL/SQL procedure that returns an array of strings:

**Example 4: PL/SQL Procedure Returning an Array of Strings**

```
MachineName=skini-pc
Port=1521
```

```
SID=o817
UserName=scott
password=tiger
STATEMENT=begin skini_junk.newproc(:1,:2); end;
NUMROWS=30
SQLINPARAM1=SYSTEM
SQLOUTPARAMPOS=2
SQLOUTPARAMTYPE=my_string_array
```

The following example specifies a PL/SQL package that returns an array of structures:

**Example 5: PL/SQL Package Returning an Array of Structures**

```
MachineName=skini-pc
Port=1521
SID=o817
UserName=scott
password=tiger
STATEMENT=begin :1 := skini_junk.func2(:2,:3,:4,:5,:6); end;
NUMROWS=30
SQLINPARAM2=somename
SQLINPARAM3=someplace
SQLINPARAM4=someanimal
SQLINPARAM5=something
SQLINPARAM6=22
SQLOUTPARAMPOS=1
SQLOUTPARAMTYPE=my_struct_array
```

The following example provides the PL/SQL used in the previous examples for reference.

**Example 6: PL/SQL Used in Examples**

```
create or replace type my_type as Object (
    name varchar2(128),
    place varchar2(128),
    animal integer,
    thing number,
    thing2 number);
/
create or replace type my_struct_array as table of my_type;
/

create or replace type my_string_array as table of varchar2(3000);
/

create or replace type my_int_array as table of integer;
/

create or replace package skini_junk as

type Jcr is ref cursor;

function func1(username in varchar2) return Jcr;
function func2(name varchar2, place varchar2, animal integer,
            thing number, thing2 number) return my_struct_array;
procedure newproc(name varchar2, outArray OUT my_string_array);
procedure newproc2(numrows in varchar2, outArray OUT my_int_array);

end skini_junk;
/
```

```
create or replace package body skini_junk as

function func1(username in varchar2) return Jcr is
cr Jcr;
begin
    open cr for select object_name, object_type, status from all_objects where
        owner=upper(username);

     return cr;
end;

function func2(name varchar2, place varchar2, animal integer,
             thing number, thing2 number) return my_struct_array IS
ret my_struct_array := my_struct_array();

begin
    ret.extend(50);

    for i in 1..50 loop
        ret(i) := my_type(name || i,
                          place || i,
                          animal+i,
                          thing+i,
                          thing2+i);
    end loop;
    return ret;
end;

procedure newproc(name varchar2, outArray OUT my_string_array) IS
begin
    outArray := my_string_array();
    outArray.extend(100);

    for i in 1..100 loop
        outArray(i) := name || i;
    end loop;
end;

procedure newproc2(numrows in varchar2, outArray OUT my_int_array) IS
begin
    outArray := my_int_array();
    outArray.extend(numrows);
    for i in 1..numrows loop
        outArray(i) := i;
    end loop;
end;

end skini_junk;
/
```

# SNMP Fetchlet

An *object identifier* (OID) corresponds to either a MIB variable instance or a MIB variable with multiple instances. Given a list of OIDs, the SNMP fetchlet polls an *SNMP agent* on a given host for corresponding instances.

**Input Parameters**

**Table 20-5    SNMP Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| hostname | String | Host name of the SNMP agent | Required. Default is "localhost" <br><br> Examples: <br> "bigip.host.example.com" <br> "148.87.10.5" |
| PORT | String | Port of the SNMP agent | Optional. Default is "161" |
| COMMUNITY | String | SNMP community string | Optional. Default is "public" |
| TIMEOUT | String | SNMP timeout. | Optional. Default is five seconds |
| OIDS | String | A list of substrings separated by delimiters. Each substring starts with an OID (in numerical dot notation), and can be optionally ended with *PlacementOID. (See notes for details.) | Required. <br> Examples: <br> "1.3.6.1.2.1.2.1.1.1.0,1.3.6.1.2.1.2.1.1.3.0,1.3.6.1.2.1.2.1.1.5.0" <br><br> "1.3.6.1.2.1.2.1.2.2.1.2 <br> 1.3.6.1.2.1.2.1.2.2.1.10 <br> 1.3.6.1.2.1.2.1.2.2.1.16" <br><br> "1.3.6.1.2.1.2.2.1.3 1.3.6.1.2.1.2.2.1.5 <br> 1.3.6.1.2.1.4.20.1.1*1.3.6.1.2.1.4.20.1.2 <br> 1.3.6.1.2.1.4.20.1.3*1.3.6.1.2.1.4.20.1.2" |
| DELIM | String | A delimiter to separate individual substrings in OIDS. | Optional; default is whitespace characters, (dot), *(star) and 0-9 (digits) cannot be used as delimiters |
| TABLE | String | Each OID in OIDS corresponds to a variable with multiple instances if this parameter is "TRUE" and to a single variable instance if it is "FALSE". | Optional; default is "FALSE" |
| PINGMODE | Boolean | Used for defining PINGMODE Response metric <br><br> If set to TRUE, then a successful GetResponse generates a single-row, single-column table with the value "1" in its cell. A timeout generates a single-row, single-column table with the value "0". <br><br> This is useful for defining a Response metric for an SNMP-based target. | Optional. Default is "FALSE" |

**Table 20-5    (Cont.) SNMP Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| IGNORE_TIME OUT_ERR_BO OLEAN | Boolea n | Specifies whether to generate an error when a non-PINGMODE invocation times out while waiting for a response.<br><br>If set to TRUE, then a non-PINGMODE invocation that times out while waiting for a response should *not* generate a metric collection error. This is reasonable behavior for targets that define a PINGMODE Response metric. If that Response metric is going to generate an availability severity when the SNMP agent stops responding, then there is no need to generate metric errors on any non-Response metrics that happen to run before the Response metric can identify the problem. | Optional. Default is "TRUE" |
| MAX_NUM_RO WS_FETCH | Integer | The maximum number of rows to be returned by a TABLE invocation.<br><br>The configuration property "SnmpTableMaxNumRowsFetch" can override the default value. | Optional. Default is 1000 |
| CONTEXT_NA ME | String | Along with CONTEXT_ENGINE_ID, these two properties specify a set of SNMPv3 credentials, which replace the community string used by SNMPv1 and SNMPv2c.<br><br>**Note**: If these two properties are specified, then the COMMUNITY and the VERSION parameters are ignored, and the sent request is an SNMPv3 request. | Optional. No default value |
| CONTEXT_ENG INE_ID | String | For information about this parameter, see the CONTEXT_NAME description. | Optional. No default value |

**Table 20-5    (Cont.) SNMP Fetchlet Input Parameters**

| Parameter | Type | Description | Use |
|---|---|---|---|
| VERSION | String | Specifies the SNMP version.<br><br>If the following occurs, then VERSION is set to "v2c", indicating an SNMPv2c request:<br><br>• disallowV1Requests is set to TRUE or hasV2Types is set to TRUE<br><br>and<br><br>• CONTEXT_NAME and CONTEXT_ENGINE_ID are not specified<br><br>If these previous conditions do not apply, then VERSION is set to "v1" indicating an SNMPv1 request. | Optional. Default is "v1" |
| disallowV1Requests | Boolean | This parameter enables the user to specify that the Management Agent should use SNMPv2c only when sending any request to a particular target | Optional. Default is "FALSE" |
| hasV2Types | Boolean | This parameter is a global-scoped property for an SNMP QueryDescriptor that includes OIDs for MIB variables whose types are 64-bit integer values. These are not representable in SNMPv1. Even if other requests for the same target instance are sent using SNMPv1, the target-type owner knows that this request must be SNMPv2c. | Optional. Default is "FALSE" |
| USE_GET_NEXT_ONLY | Boolean | If an SNMP QueryDescriptor is SNMPv2c, according to the conditions described in the VERSION description, and if TABLE is TRUE, then the multiple rows that the SNMP fetchlet returns will be fetched using the SNMPv2c GetBulk request, and not the GetNext request used in SNMPv1.<br><br>If USE_GET_NEXT_ONLY is set to TRUE, then the SNMP fetchlet returns will be fetched using GetNext requests. | Optional. Default is "FALSE |

### Error Handling

`MissingParameterException` is thrown if either host name or OIDS is not given.
`FetchletException` is thrown if TABLE is not equal to either TRUE or FALSE, an

I/O error occurs while sending or receiving SNMP messages to or from the agent, or the agent responds with an SNMP error.

**Notes**

The table returned by the fetchlet contains a column for every OID in OIDS. If input OIDs correspond to single variable instances, the table will have just one row with those instances. On the other hand, if the OIDs correspond to variables with multiple instances, each column in the table will contain instances for its OID and each row will correspond to a different *subidentifier*. (A subidentifier is an OID extension that uniquely identifies a particular variable instance for some MIB variable.) OIDS must contain either all OIDs with subidentifiers or all OIDs without the subidentifiers.

For example, to request the variable instances for the three OIDs: `sysDescr`, `sysUpTime`, and `sysName`, OIDS would have to be "1.3.6.1.2.1.2.1.1.1.0 1.3.6.1.2.1.2.1.1.3.0 1.3.6.1.2.1.2.1.1.5.0". In this case, all OIDs contain the instance subidentifier, ".0". The return table appears as follows (the actual values may be different):

**Figure 20-4    SNMP Fetchlet**



Alternatively, assume that some MIB contains the following 3 columns and 4 instances:

**Figure 20-5    SNMP Fetchlet: Columns 3 and 4 Content**



To construct a table with 3 columns corresponding to `ifDescr`, `ifInOctets`, and `ifOutOctets`, OIDS would be defined as follows

"1.3.6.1.2.1.2.1.2.2.1.2 1.3.6.1.2.1.2.1.2.2.1.10 1.3.6.1.2.1.2.1.2.2.1.16"

The fetchlet returns the following:

**Figure 20-6    SNMP Fetchlet:ifDescr, ifInOctets, and ifOutOctets OIDS**

| | | |
|---|---|---|
| wx1 | 26150844 | 29368527 |
| wx2 | 2763185941 | 3023812977 |
| wx3 | 123615396 | 2055140730 |
| wx 4 | 2068257723 | 3212899913 |

The rows correspond to subidentifiers 1,2,3,4 respectively.

Any OID in OIDS can be appended with another *placement* OID. The variable instances for the placement OID do not appear in the returned table. Instead, they determine the place for the variable instances of the original OID within a column. In particular, for every variable instance I with subidentifier S in the set of instances for the original OID, (a) there must exist a variable instance X with subidentifier S in the set of instances corresponding to the placement OID, and (b) X is used as the subidentifier for the instance I.

For example, consider a MIB containing the following 3 columns, each with 4 variable instances:

**Figure 20-7    SNMP Fetchlet: MIB Content with 4 Variable Instances**

| ifDescr (network interface description) | ipAdEntNetMask (netmask) | ipAdEntIfIndex (network interface index) |
|---|---|---|
| OID: 1.3.6.1.2.1.2.1.2.2.1.2 | OID: 1.3.6.1.2.1.2.1.4.20.1.3 | OID: 1.3.6.1.2.1.2.1.4.20.1.2 |
| subidentifier : variable instance | subidentifier : variable instance | subidentifier : variable instance |
| 1: wx1 | IP1: 255.255.255.0 | IP1: 3 |
| 2: wx2 | IP2: 255.255.128.0 | IP2: 1 |
| 3: wx3 | IP3: 255.255.255.240 | IP3: 4 |
| 4: wx 4 | IP4: 255.255.254.0 | IP4: 2 |

To construct a table containing `ifDescr` and `ipAdEntNetMask`, OID of `ipAdEntIfIndex` would have to be used as the placement OID to "align" the columns. Thus, the OIDS input to the fetchlet would be "1.3.6.1.2.1.2.1.2.2.1.2 1.3.6.1.2.1.2.1.4.20.1.3*1.3.6.1.2.1.2.1.4.20.1.2". The fetchlet output will be as follows:

**Figure 20-8    SNMP Fetchlet: Table Containing ifDescr and ipAdEntNetMask**

| | |
|---|---|
| wx1 | 255.255.128.0 |
| wx2 | 255.255.254.0 |
| wx3 | 255.255.255.0 |
| wx 4 | 255.255.255.240 |

If OIDS were "1.3.6.1.2.1.2.1.2.2.1.2 1.3.6.1.2.1.2.1.4.20.1.3" for the previous example, the output would be as follows:

**Figure 20-9    SNMP Fetchlet: Alternate OID**



# HTTP Data Fetchlets

The HTTP data fetchlets obtain the contents of a URL and returns the contents of the URL as data. Three fetchlets are available:

- URL Fetchlet
- URL Lines
- URL Lines Token

## URL Fetchlet (raw)

The URL fetchlet gets the contents of a given URL and returns the contents of the URL in a single cell table.

To get the output of a URL tokenized by lines and each line tokenized by a given delimiter, see the URL Line Token fetchlet.

**Input Parameters**

**Table 20-6    URL Fetchlet Input Parameters**

| Name | Description | Use |
|------|-------------|-----|
| url | URL to retrieve the contents of | required |
| proxyHost | proxy host through which to make the URL connection. | optional |
| proxyPort | proxy port through which to make the URL connection. | optional |

**Example**

Take the following URL:

```
http://localhost/nhcities.txt
```

It has the following contents:

*Line 1*: Nashua, Keene,

*Line 2*: Concord

*Line 3*: , Conway, Manchester, Milford, Brookline,

*Line 4*:

*Line 5*: Hollis, Meredith

Now run the URL fetchlet with the given URL. The fetchlet returns the following one-by-one table:

**Figure 20-10    URL Fetchet Output**

```
Nashua, Keene,
Concord
, Conway, Manchester, Milford, Brookline,

Hollis, Meredith
```

The raw contents of the URL is returned.

**Error Handling**

MissingParameterException if URL parameter is missing. FetchletException if the URL is malformed or an I/O error occurs in retrieving the content of the URL.

## URL Lines Fetchlet (split into lines)

The URL fetchlet gets the contents of a given URL and tokenizes the contents of the URL. The output is tokenized by lines. The fetchlet returns the tokens in a single column table. The nth row in the table represents the nth line of the URL contents.

> **Note:**
>
> To get the raw, untokenized contents of a URL, see the URL fetchlet. To get the contents of a URL tokenized by lines and each line tokenized by a given delimiter, see the URL Line Token fetchlet.

**Table 20-7    URL Lines Fetchlet Input Parameters**

| Name | Description | Use |
|---|---|---|
| url | URL to retrieve the contents of | required |
| proxyHost | proxy host through which to make the URL connection. | optional |
| proxyPort | proxy port through which to make the URL connection. | optional |

**Table 20-7 (Cont.) URL Lines Fetchlet Input Parameters**

| Name | Description | Use |
|---|---|---|
| startsWith | only lines starting with this string are included in the result | optional; default = "" (all lines are included) |

**Example**

Take the following URL:

```
http://localhost/nhcities.txt
```

It has the following contents:

*Line 1*: Nashua, Keene,

*Line 2*: Concord

*Line 3*: , Conway, Manchester, Milford, Brookline,

*Line 4*:

*Line 5*: Hollis, Meredith


Now run the URL fetchlet with the given URL.

The fetchlet returns the following table:

**Figure 20-11 URL LInes Fetchlet Output**



**Error Handling**

MissingParameterException if URL parameter is missing.

FetchletException if the URL is malformed or an I/O error occurs in retrieving the content of the URL.

**Notes**

Lines are tokenized using "\n".

# URL Line Token Fetchlet (tokenized lines)

The URL fetchlet gets the contents of a given URL and tokenizes the contents of the URL. The output is tokenized first by lines, and then each line is tokenized by a

given delimiter set. The fetchlet returns the tokens in a table. The *n*th row in the table represents the *n*th line of the URL content. The *n*th column in the table represents the *n*th token in a line as determined by the given delimiter set.

To get the raw, untokenized contents of a URL, see the URL fetchlet.

**Table 20-8    URL Line Token Fetchlet Input Parameters**

| Name | Description | Use |
| --- | --- | --- |
| url | URL to retrieve the contents of | required |
| delimiter | set of characters that act as delimiters to tokenize the lines | optional; default = "" (just breaks output into lines) |
| proxyHost | proxy host through which to make the URL connection. | optional |
| proxyPort | proxy port through which to make the URL connection. | optional |
| startsWith | only lines starting with this string are included in the result | optional; default = "" (all lines are included) |

**Example**

Take the following URL:

```
http://localhost/nhcities.txt
```

It has the following contents:

Line 1: Nashua, Keene,

Line 2: Concord

Line 3: , Conway, Manchester, Milford, Brookline,

Line 4:

Line 5: Hollis, Meredith

Now run the URL fetchlet with the given URL and a single character "," for the delimiter.

The fetchlet returns the following table:

**Figure 20-12    URL Token Lines Output**



**Error Handling**

MissingParameterException if URL parameter is missing.

FetchletException if the URL is malformed or an I/O error occurs in retrieving the content of the URL.

**Notes**

Lines are tokenized using "\n".

The delimiter can be a single character or a set of characters. For example, it can be "|+_", if the line should be broken up by pipes, pluses, and underscores. If two or more delimiters are together in the output text, such as "||" or "+|+", then it is as if there are empty string tokens between them. These empty strings get columns in the result table. It is NOT considered that there are empty strings preceding a delimiter that starts a line or following a delimiter that ends a line.

In order to express non-printable characters in the delimiter set (such as tabs) in XML, use "&#xHH;" where H is the hexadecimal identifier for the character.

# URLXML Fetchlet

The URL XML fetchlet obtains the XML content of a given URL, and extracts information based on a given pattern. A pattern is a list of "chunks" of XML to match against. The return table is a table with a column for each grabber (*) in the pattern in order and a row each time the pattern chunk matches in the XML content.

**Input Parameters**

**Table 20-9    URLXML Fetchlet Input Parameters**

| Name | Description | Use |
|---|---|---|
| url | URL to retrieve the contents of | Required. |
| pattern | The pattern used to extract information from XML; this is a list of XML chunks that that is compared against the XML content of the URL. Each chunk contains one or more "grabbers" (*) in the text portion of the elements that define what should the flattened into text and extracted. | Required. |
| proxyHost | The proxy host through which to make the URL connection. | Optional. |
| proxyPort | The proxy port through which to make the URL connection. | Optional. |
| ignoreDtd | If set to TRUE, specifies that the DTD file referenced by the content XML should be ignored. This is useful in cases where the DTD file cannot be accessed. | Optional. |
| generateKey | If set to true, a unique key will be generated for each row. The key will occupy the first column of the result, and will be numeric. | Optional. |
| throwConnException | If set to TRUE, a java.net.ConnectException will be thrown. Otherwise, it will be caught and an empty result set will be returned. Setting this property to FALSE provides behavior which is consistent with the DMSFetchlet. | Optional. The default value is TRUE. |

**Example**

Take the following URL:

```
http://localhost/urlxmltestfile.xml
```

It has the following content:

```
<?xml version="1.0"?>
<testfile>
    <test>Simple text</test>
    <test><level>A little more complex</level></test>
    <test></test>
    <notatest></notatest>
    <test>Yet more complexity<level>Even a little more complex</level>Will it
ever stop?</test>
    <test1>must match<level>extract me!</level></test1>
    <test1>must match here<level>extract me, too!</level></test1>
</testfile>
```

Running the URL XML fetchlet with the given URL and the pattern:

```
<testfile><test>*<level>*</level></test></testfile>
```

returns the following table:

**Figure 20-13    URL XML Fetchlet Output**

| A little more complex | A little more complex |
|---|---|
| Yet more complexityEven a little more complexWill it ever stop? | Even a little more complex |

**Error Handling**

MissingParameterException if URL or pattern parameters are missing.

A FetchletException is generated if:

- The URL is malformed.

- An I/O error occurs in retrieving the content of the URL.

- The URL contents or pattern contains invalid XML.

**Notes**

Setting the proxy host and/or port changes these settings for the java.net package for the whole Java environment and is not thread-safe if the proxy settings are changing.

# URL Timing Fetchlet

The URL Timing fetchlet gets the contents of a given URL, timing not only the base page source but any frames or images in the page as well.

**Input Parameters**

**Table 20-10    URL Timing Fetchlet Input Parameters**

| Parameter | Description | Use |
|---|---|---|
| url# | URL(s) to download. "url0" is required but any number of URLs can be specified beyond url0 that following the convention: url0, url1, url2, url3. | Required. |
| proxy_host | Proxy host used to make a URL connection. | Optional. Specifies the proxy to be used for accessing URLs. If the proxy_host_override value is provided, then that value will be used instead. |
| proxy_port | Port used by the proxy host used make the URL connection. | Optional. |
| dont_proxy_for | Domains for which the proxy will not be used. | Optional. For example, .us.example. com, .uk.example.com |
| use_proxy | When used in conjunction with the proxy override input parameters, use_proxy specifies a proxy to be used in lieu of the original proxy. When set to false without the proxy override parameters set, no proxy is used. | Optional. Parameter can be set to true or false. |
| proxy_host_overri de | Alternate proxy host used to make the URL connection. | Optional. Overrides proxy_host. |
| proxy_port_overri de | Alternate proxy port used to make the URL connection. | Optional. Overrides proxy_port. |
| dont_proxy_overri de | Do not use the proxy for domains. | Optional. Parameter can be set to true or false. |
| internet_cert_loc | Path pointing to the location of a certificate to be used to access a secure (HTTPS) URL. | Optional. |
| auth_realm | Realm for the Basic Authentication log on. If the realm is not specified for the authentication, authentication does not occur and the download of the page fails with a 401 response code. | Optional. |
| auth_user | User name for Basic Authentication. | Optional. |
| auth_password | Password for Basic Authentication. | Optional. |
| retries | Number of times to retry the url if it initially fails. | Optional. Default = 1 |
| connection_timeo ut | Wait time (in milliseconds) allowed to establish a connection to a server. This time also includes time required for name resolution. | Optional. Default= 60000 milliseconds (1 minute) |
| read_timeout | Idle time in the read waiting for the server to respond. For example, if no data is received from the server during the specified timeout period, the operation is considered failure. | Optional. Default = 12000 milliseconds (2 minutes) |
| timeout | Number of milliseconds after which the page download is considered a failure. This will detect if the site is functional but is extremely slow. | Optional. Default = 300000 ms (5 minutes) |

**Table 20-10    (Cont.) URL Timing Fetchlet Input Parameters**

| Parameter | Description | Use |
|---|---|---|
| status_comparator | When collating the rows to make a single row, the status_comparator parameter will indicate whether all URLs should have been a success (and) or any URLs should have been a success (or). | Optional. Default = and |
| cache | Indicates whether to use a cache when accessing an URL. Set the parameter to "n" to specify that no cache be used. | Optional. Default = yNote: The scope of the cache is per request. There is no persistent cache across multiple get metric requests. |
| output_format | Specifies the output format to be used: summary, detailed, repeat_column. For more information on output formats, see Metric Columns and Output Modes. | Required. summary : gives a default set of metrics in a single row for all urlsdetailed: gives a default set of metrics for each url.repeat_column : gives a single row of metric with timing for each of the url. |
| metrics | Specifies which metric columns need to be returned. For more information on metrics columns returned for each output format, see Table 20-12 | Optional. Allows you to specify of what needs to be returned from the fetchlet and in which order.Example: status, status_description, total_response_time |

**Metric Columns and Output Modes**

The format of information and specific metric information returned are controlled by the "output_format" and "metrics" input parameters. Table 20-11 lists the format categories and the metrics (columns) returned by each. For a description of available metric columns, see Table 20-12

**Table 20-11    URLTIMING Fetchlet: Output Formats**

| Output Format | Description | Metric Columns |
|---|---|---|
| summary | Returns a default set of metrics in a single row for all URLs<br><br>If the metrics input parameter is specified, then only the columns specified will be returned. | computed_response_time, status, status_description, dns_time, connect_time, redirect_time, first_byte_time, html_time, content_time, total_response_time, rate, max_response_time, avg_response_time, avg_connect_time, avg_first_byte_time, broken_count, broken_content |

**Table 20-11    (Cont.) URLTIMING Fetchlet: Output Formats**

| Output Format | Description | Metric Columns |
|---|---|---|
| detailed | Returns a default set of metrics for each url.<br><br>If the metrics input parameter is specified, then only the columns specified will be returned. | url, computed_response_time, status, status_description, dns_time, connect_time, redirect_time, first_byte_time, html_time, content_time, total_response_time, rate, redirect_count, html_bytes, content_bytes, total_bytes, avg_connect_time, avg_first_byte_time, broken_count, broken_details |
| repeat_column | Returns a single row of metrics with timing for each of the URLs.<br><br>If the metrics input parameter is specified, then those columns will be returned for each URL followed by overall status and status_description. (Note the output will always be single row). | total_response_time repeated for each URL followed by overall status and status_description. |

**Metric Columns**

Table 20-12 shows the metric columns returned by the URLTIMING fetchlet.

**Table 20-12    URLTIMING Fetchlet: Metric Columns**

| Column Name | Description |
|---|---|
| status | The overall status of all URLs. By default AND is used to compute overall status but this can be changed using the status_comparator input parameter. |
| connect_time | The time to connect to the server and send the request. |
| first_byte_time | Time taken between sending the request and reading the first byte from the response. |
| total_response_time | Time taken for fetching ALL urls and associated content (gif, css, javascript, and so on). |
| max_response_time | Also referred as Slowest page time. This the time taken by the slowest URL. |
| avg_response_time | Average response time for URL. Computed as total response time / number of pages (urls). |
| rate | Kilo Bytes per second. This is computed by total bytes received / total time taken to receive them. |
| html_time | Total time taken to download the html part of all pages. This time excludes time to fetch images and other contents. (Includes time to fetch frame html). |
| content_time | Time taken to download the page content (gif, javascripts, css, and so on). |

**Table 20-12    (Cont.) URLTIMING Fetchlet: Metric Columns**

| Column Name | Description |
| --- | --- |
| redirect_time | Total time taken for all redirects occurring while fetching the set of urls specified. |
| redirect_count | Number of redirects. |
| total_bytes | Total number of bytes. |
| html_bytes | Total number of HTML bytes. (Includes bytes for frame html). |
| content_bytes | Total number of content bytes. |
| status_description | This is present only when the status is down. Corresponds to HTTP Status description. |
| request_count | Number of request made. (Includes all html as well as content requests). |
| broken_count | Number errors while fetching images or other content elements. |
| broken_details | List of images or other content elements that could not be fetched. This has format of url[broken list], url[broken list... |
| computed_response_time | This time approximates the time it would have taken for a client (like browser) to fetch all the pages in the transaction. This number is computed as if the contents of every page (gifs, css, and so on) were fetched using multiple threads. |
| avg_connect_time | Total connect_time / total number of connections made. |
| avg_first_byte_time | Total First Byte Time / Number of requests made (either to fetch HTML or content). |
| dns_time | Time to resolve host name (not implemented, always returns zero). |
| url | Returns the url, can only be used in 'detailed' output_format. |

**Example**

Take the following URL:

```
url0=http://www.example.com/
```

With the input parameter output_format=summary, the fetchlet returns the following table (minus the headers on the columns):

**Figure 20-14    Summary Output Format**

| computed response time | status | status description | dns time | connect time | redirect time | first byte time | html time | content time | total response time | rate (Kbytes per second) | max response time | avg response time | avg connect time | avg first byte time | broken count | broken content |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 540 | 1 | | 0 | 548 | 0 | 149.0 | 1.0 | 7.0 | 705 | 95.16 | 705 | 705 | 54.80 | 14.90 | 0 | |

With output_format = summary and metrics = total_response_time, status, status_description the fetchlet returns the following table (minus the headers on the columns):

**Figure 20-15    Summary Output Format with Specified Metric Columns**

| total response time | Status | Status description |
|---|---|---|
| 705.0 | 1 | |

With output_format = summary and metrics = total_response_time, status, status_description the fetchlet returns the following table (minus the headers on the columns) and the server is giving error:

**Figure 20-16    Summary Output Format with Specified Metric Columns and Internal Server Error**

| total response time | status | status description |
|---|---|---|
| | 0 | Internal Server Error -- http://www.example.com |

Take the following URL:

```
url0=http://www.example.com/
url1=http://nedc.us.example.com/
```

With the output_format=summary, the fetchlet returns the following table (minus the headers on the columns). Here the numbers are time taken for fetching both the urls.

**Figure 20-17    Summary Output Format for Two URLs**

| computed response time | status | status description | dns time | connect time | redirect time | first byte time | html time | content time | total response time | rate (Kbytes per second) | max response time | avg response time | avg connect time | avg first byte time | broken count | broken content |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4344 | 1 | | 0 | 603 | 0 | 7277 | 438 | 318.0 | 8636 | 16.92 | 8098 | 4318 | 22.33 | 269.52 | 0 | |

With the output_format=detailed, the fetchlet returns the following table (minus the headers on the columns):

**Figure 20-18    Detailed Output for Two URLs**

| url | uri | computed response time | status | status description | dns time | connect time | redirect time | first byte time | html time | content time | total response time | rate | redirect count | html bytes | content bytes | total bytes | avg connect time | avg first byte time | broken count | broken content |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| http://www.example.com | not supported | 531.0 | 1 | | | 548 | 0 | 149.0 | 1 | 7 | 705 | 95.16 | 0 | 18207 | 48883 | 67090 | 54.80 | 14.90 | 0 | |
| http://nedc.us.example.com | not supported | 4424.0 | 1 | | | 480 | 0 | 6227.0 | 442 | 230 | 7379 | 10.71 | 0 | 24627 | 54427 | 79054 | 28.24 | 366.29 | 0 | |

With the output_format=repeat_column, the fetchlet returns the following table (minus the headers on the columns):

**Figure 20-19    Repeat Column Output Format**



**Error Handling**

Metric error if the URL parameter is missing, malformed, or if the metric cannot be computed.

**Notes**

The time required to perform a retry will be added on to the total time of the page. For example, if two retries are performed and then a success occurs, the total page time will be the time of the page that succeeded plus the time it took for the two retries to fail.

**Proposed usage**:

- For basic monitoring:

    Use url0=<URL to be monitored> , output_mode=summary and specify metrics=status, computed_response_time, status_description

- For getting all columns:

    Use url0=<url to be monitored> , output_mode=summary

# Dynamic Monitoring Service (DMS) Fetchlet

The Dynamic Monitoring Service (DMS) fetchlet contacts an Application Server (AS) and then collects the metrics instrumented by the DMS.

The DMS allows application and system developers to measure and export customized, component-specific performance metrics. The Oracle Management Agent allows software components to import runtime performance data into Oracle Enterprise Manager Cloud Control.

The DMS fetchlet is an Oracle Management Agent plug-in module that allows the Management Agent to import the performance data that is exported by the DMS. Using the DMS fetchlet, any component that is instrumented using DMS API calls may share its performance data with Enterprise Manager Cloud Control.

## Advantages to Using DMS for Oracle Management Agent Integration

With DMS, a component can insulate itself from the operational details of the Management Agent. A component would not need to deploy (or maintain) its own fetchlet or deploy (or maintain) a Tcl script or shell script to plug into one of the existing fetchlets. A component would not need to devise its own new way of measuring or exporting performance metrics. Performance metrics can be measured and reported in a consistent way across components. The DMS fetchlet contacts the remote DMS runtime directly with no need for forking shell scripts or Tcl scripts. Most importantly, DMS automatically produces the long, complicated metadata document for you and thereby saves many hours of tedious and error-prone hand editing.

**Input Parameters**

**Table 20-13    DMS Fetchlet Input Parameters**

| Name | Type | Description | Use |
|---|---|---|---|
| oraclehome | String | Top directory under which the monitored IAS instance is installed. It is used only for monitoring local IAS processes. For monitoring remote IAS processes, users should give it an empty value and specify property "opmnremoteport" and/or "machine" instead. | Required. Example: "/private/oracle/ias" |
| version | String | AS Version number of the target. It is used to distinguish the version of monitored AS instance. | Optional Example: "9.0.4" |
| opmnport | Integer | Oracle Process Monitoring and Notification (OPMN) port. It is used primarily for monitoring remote AS processes. It should be specified together with property "machine". If it is present and valid , property "oraclehome" and "httpport" are ignored. | Optional Example: "6200" |
| httpport | Integer | HTTP port is used primarily for monitoring stand-alone processes. It should be specified together with property "machine". It will be ignored, if property "opmnport" is present. If it is present and valid, property "oraclehome" is ignored. | Optional Example: "7777" |
| machine | String | Host name where the Internet Application Server (AS) instance runs. It should be specified together with property "opmnport". If it is not present, the local host is assumed. | Optional Example: "my-sun.us.example.com" |
| metric | String | Name of the table-type metric. | Required Example: "Servlets" |
| columnOrder | String | A list of metric column names separated by ";". The column names must be specified in same order as they appear in the target type metadata file. Do not include "name", "host", "process" and "fullname" columns. | Required Example: "processTimes;totalReque st;requestRate" |
| usecache | String | Whether to cache this metric. The legal values are "true", "false" and "refreshall" with "true" being the default. The "refreshall" value tells the DMS to delete its cache data and retrieve the most recent data from all targets. | Optional. Example: "false" Setting "usecache" to "false" will bypass DMS caching |

**Table 20-13 (Cont.) DMS Fetchlet Input Parameters**

| Name | Type | Description | Use |
|------|------|-------------|-----|
| proxyHost | String | Proxy host through which to make the HTTP connection | Optional<br>Example:<br>"proxy.us.example.com" |
| proxyPort | Integer | Proxy port through which to make the HTTP connection | Optional<br>Example:<br>"80" |
| dontProxyFor | String | Domains for which the proxy will not be used. | Optional<br>Example:<br>".us.example.com" or "18.219.0" |
| useDefaultProxy | String | When used in conjunction with the proxy override parameters, this variable specifies a proxy other than the original one. When set to false without the proxy override parameters set, no proxy at all is used. | Optional<br>Example:<br>"true" or "false" |
| proxyHostOverride | String | proxy host through which to make the HTTP connection | Optional<br>Example:<br>"www-proxy.us.example.com" |
| proxyPortOverride | Integer | proxy port through which to make the HTTP connection | Optional<br>Example:<br>"80" |
| authrealm | String | Realm for the Basic Authentication logon. If the realm is not specified for the authentication, authentication does not occur and the download of the page fails with a 401 response code. | Optional<br>Example:<br>"Please input your flex account login:" |
| authuser | String | Username for Basic Authentication | Optional<br>"superuser" |
| authpwd | String | Password for Basic Authentication | Optional<br>Example:<br>"welcome" |

**Error Handling**

The DMS fetchlet throws MissingParameterException if any of the properties "oraclehome", "metric", "columnOrder", "opmnport", or "httpport" is missing. It throws FetchletException if any of the ports given is not valid.

**Notes**

The first four columns of the metric table returned are always column "name", "fullname", "host" and "process". Therefore, do not include them in columnOrder string. Property "machine" should be specified together with either properties "opmnport" or "httpport". In this case, the property "oraclehome" is ignored.

# DMS Fetchlet/Oracle Management Agent Integration Instructions

DMS has been used in several components (such as Apache, JServ, OSE, and Portal) to provide a consistent performance monitoring infrastructure for Oracle 9*i* Application Server. The Sensors are easy to use and save most of the work related to performance measurement because they hide most of the details related to timing, counting, and categorization. Finally, DMS hides many Management Agent details from component developers and much of the Management Agent integration effort.

## Integrating DMS Data with the Management Agent

As mentioned earlier, DMS allows application and system developers to measure and export customized, component-specific performance metrics. The Oracle Management Agent enables software components to import runtime performance data into Enterprise Manager Cloud Control. This section describes how to integrate DMS performance metrics with the Management Agent.

**Step 1: Install AS**

**Step 2: Install Enterprise Manager Cloud Control**

**Step 3: Instrument your Component with DMS**

To enable DMS metrics for Enterprise Manager Cloud Control, you must follow two additional rules:

- **Rule 1**: *All Nouns exported to the Management Agent must have types* Noun types can be set either by specifying the "type" parameter in the Noun.create() methods or by using the Noun.setType(String) method. The idea is that every Noun type will be converted automatically to a Management Repository table. Every Noun of a given type will become a row in the type's corresponding Management Repository table. The metrics contained by a Noun become columns in the Management Repository table metric. Any Noun without a type will not be exported to Management Agent.

- **Rule 2**: *All Nouns of a given type must contain a consistent set of Sensor names* Because the metrics contained by a Noun become columns in a management repository table, you must make sure that all Nouns of a given type contain the same Sensors. This ensures that each row of the corresponding Management Repository table has the same set of columns. DMS does not check this constraint for you.

For example, the following Java snippet shows how to create typed Nouns that contain a consistent set of Sensors. DMS will automatically convert these into a Management Repository table named "MyType":

```
/* first create the nouns*/
Noun n1 = Noun.create("/myExample/myComponent/noun1", "MyType");
Noun n2 = Noun.create("/myExample/myComponent/noun2", "MyType");

/* next, create the Sensors */
PhaseEvent pe1 = PhaseEvent.create(n1, "criticalPhase", "a critical interval");
PhaseEvent pe2 = PhaseEvent.create(n2, "criticalPhase", "a critical interval");
Event e1 = Event.create(n1, "importantEvt", "an important event");
Event e2 = Event.create(n2, "importantEvt", "an important event");

/* here is a third set that shows the use of Noun.setType(String) */
PhaseEvent pe3 = PhaseEvent.create(
```

```
                                  "/myExample/myComponent/noun3/criticalPhase",
                                  "a critical interval");
Event e3 = PhaseEvent.create(
                                  "/myExample/myComponent/noun3/importantEvt",
                                  "an important event");
Noun n3 = Noun.get("/myExample/myComponent/noun3");
n3.setType("MyType");
```

For this example, the "MyType" table will contain three rows and four columns. Besides the columns corresponding to the two Sensors, there will be a "name" column and a "path" column that will contain the DMS path name including the process name and "/myExample/myCom...".

If these Nouns/Sensors are created in several servlet engines within the AS site, then the AggreSpy will find each of the servlet engines and will aggregate all of the Nouns/Sensors into a single MyType table.

**Step 4: Generate your Target Metadata Document**

You can generate the Target Metadata Document using your browser. Point your browser to your AS site that you want to monitor using the following URL:

```
http://YOUR_AS_HOST:YOUR_AS_PORT/YOUR_SERVLET_PATH/AggreSpy?format=targetmetadata
```

You should use the actual host, port and servlet path of your AS installation in the above URL. The servlet path usually defaults to "servlet". The XML document you get is the Target Metadata Document for your AS site. The first comment of the XML document explains where you can obtain the Target Metadata Document and instructions telling you what needs to be done to this document.

**Step 5: Install the Target Metadata Document**

Follow the steps described in the first comment of the XML document. Save the XML document to a file called "oracle_dms.xml" under the "metadata" directory of your Enterprise Manager installation (OMS_ORACLE_HOME/sysman/admin/metadata/). If you want to monitor a subset of the metrics or merge the metrics with the ones in the existing "oracle_dms.xml" file, you should save this new definition to a separate file called target_name.xml. You will also need to change the Target Type entry in the generated metadata document.

Next, you should add the target instance information of your AS site to your "targets.xml" file residing under the top directory of your Enterprise Manager installation. You can find a block of XML tags in the comment you read. They look like:

```
<Target Type='oracle_dms' NAME='DMS_YOUR-IAS-HOST_YOUR-IAS-PORT' VERSION='2.0'>
 <Property NAME='host' VALUE='YOUR_IAS_HOST' />
 <Property NAME='port' VALUE='YOUR_IAS_PORT' />
 <Property NAME='dmsPath' VALUE='YOUR_SERVLET_PATH' />
</Target>
```

Copy this block and paste it to the targets.xml file between <targets> and </targets> tags.

Finally, to add the new target metadata file and target instance information from the targets.xml file to Enterprise Manager Cloud Control, you must run the following command:

```
>$ORACLE_HOME/bin/emctl reload
```

**Step 6: View Your Metrics**

You are ready to view your metrics using Enterprise Manager's Metric Browser. First, make sure that AS and your component are still running. Next, restart the Oracle Management Agent. Finally, point your browser to your Management Agent installation using the following URL:

```
http://YOUR_AGENT_HOST:YOUR_AGENT_PORT/emd/browser/main
```

The Management Agent port information can be found in the $AGENT_HOME/sysman/config/emd.properties file at the EMD_URL line.

You should use the actual host and port of your Management Agent installation in the above URL. You will find your AS site listed as the target "DMS_YOUR-AS-HOST_YOUR-AS-PORT". If you click the link, you will see a list of metric IDs. You can browse your metrics by clicking on the respective metric IDs.

# JDBC Fetchlet

Call-level interfaces such as JDBC permit external access to SQL database manipulation and update commands. The Java Database Connectivity (JDBC) fetchlet allows you to execute common JDBC commands and obtain their response time for any type of database.

**Input Parameters**

**Table 20-14    JDBC Fetchlet Input Parameters**

| Name | Description | Use |
|---|---|---|
| Transaction Name | (Standard) | Required. |
| Beacon Name | (Standard) | Required. |
| Connect String | Connection string provided by the user. The Connect String must comply with the URL format specified by the vendor of the database to which the user is trying to connect. | Required. |
| | Examples: | |
| | Format required by Oracle: | |
| | `jdbc:oracle:thin:@hostname:port` | |
| | Format required by MySQL: | |
| | `jdbc:mysql://hostname:port` | |

**Table 20-14    (Cont.) JDBC Fetchlet Input Parameters**

| Name | Description | Use |
|------|-------------|-----|
| Class Name String | The driver class name to be used for connections.<br><br>Example:<br><br>oracle.jdbc.driver.OracleDriver<br><br>You have two options for configuring the Agent to use the .jar file containing the driver:<br><br>1. Place the .jar file in $JAVA_HOME/jre/lib/ext. CLASSPATH does not need to be modified.<br><br>2. Place the .jar file anywhere and update CLASSPATH in emd.properties file with the path to jar. Bounce Agent. This should be scripted and be transparent to user. | Required. |
| Username | User name to be used when connecting to the database. | Required. |
| Password | Password to be used when connecting to the database. | Required. |
| Role | User Role | Required. |
| Statement | SQL statement to be executed. Use of PL/SQL is possible by using prepareCall() API. | Required. |

**Table 20-15    Metric Columns Collected**

| Column | Description |
|--------|-------------|
| Status | Status of the test. Status is 'down' if there is a SQLException generated by the fetchlet. |
| Total Time | Time required for the fetchlet to execute the test. |
| Connect Time | Time required for DriverManager.getConnection() to complete. |
| Prepare Time | Time required for conn.prepareStatement() to complete. |
| Execute Time | Time required for stmt.executeQuery() to complete. |
| Fetch Time | Time required for while(rs.next()) { rs.getRow() } to complete. |
| Close Time | Time required for closing resultset, statement, connection to complete. |
| Number of rows | Number of rows fetched. |
| Total time per row | |
| Fetch time per row | |

**Example: Properties Passed to JDBC Fetchlet**

The following example provides the properties passed to the JDBC fetchlet when invoked.

```
<QueryDescriptor FETCHLET_ID="JDBC">
<Property NAME="TxnName" SCOPE="GLOBAL">TxnName</Property>
<Property NAME="BeaconName" SCOPE="GLOBAL">BeaconName</Property>
```

```
<Property NAME="connstring" SCOPE="INSTANCE">connString</Property>
<Property NAME="username" SCOPE="INSTANCE">username</Property>
<Property NAME="password" SCOPE="INSTANCE">password</Property>
<Property NAME="statement" SCOPE="GLOBAL">select * from user_tables</Property>
<Property NAME="classstring" SCOPE="GLOBAL">oracle.jdbc.none</Property>
<Property NAME="role" SCOPE="GLOBAL" OPTIONAL="TRUE">DBA</Property>
<Property NAME="useconnpool" SCOPE="GLOBAL" OPTIONAL="TRUE">FALSE</Property>
<Property NAME="GetTimingData" SCOPE="GLOBAL">TRUE</Property>
</QueryDescriptor>
```

# WBEM Fetchlet

The WBEM fetchlet accesses a CIMOM and retrieves requested information using the specified CIM class. The CIM class is mapped to a Management Repository table metric. The name of the CIM class is the name of the table metric that is returned, and the properties defined for the CIM class are used to name the table columns for the metric. The properties of interest must be specified during metric definition.

The fetchlet returns the instances that have been instantiated for the CIM class as rows of the Management Repository table metric.

**Input Parameters**

**Table 20-16    WBEM Fetchlet Input Parameters**

| Name | Type | Description | Use |
|------|------|-------------|-----|
| hostname | String | Host name of the CIMOM | Optional; default is "localhost" |
| port | Integer | Port for the CIMOM | Optional; default is 5988 |
| namespace | String | CIM Namespace | Optional; default is "root/cimv2" |
| username | String | User name to use for CIMOM authorization on the host where the CIMOM is running | Required |
| password | String | Password to use for CIMOM authorization on the host where the CIMOM is running | Required |
| CIMclassname | String | Name of the CIM class whose instances will be returned | Required for all operations except STATUS. STATUS operations just check whether the CIMOM is running, so a class name is not needed. |
| operation | String | Operation to be performed. Supported operations include COUNT, which returns a count of the number of instances in the class, VALUES, which returns the values of the specified properties for each instance of the class, or STATUS, which provides status information about the CIMOM. | Optional, default is VALUES |

**Table 20-16    (Cont.) WBEM Fetchlet Input Parameters**

| Name | Type | Description | Use |
|------|------|-------------|-----|
| properties | String | The property names from the CIM class definition that we are interested in collecting. | Required for VALUES operation. If the operation is VALUES, we can have 1 to N of these, separated by a semicolon. If the operation is VALUES, and no properties are provided, an error is returned. Properties are handed to the EMD in the order that they are specified. |

**Error Handling**

The following types of errors have been identified for the WBEM fetchlet.

**MissingParameterException occurs when**:

- No CIM Class parameters match.

**Fetchlet exception occurs when**:

- The class name is not found in the CIMOM namespace.

- The namespace is not found.

- The connection to the CIMOM does not have valid credentials.

- The connection to the CIMOM failed because the CIMOM was not running.

- The CIM class property does not exist

- An unsupported operation was specified

- No properties were specified.

**Notes**

Ports: Some CIMOM client interfaces expose the port that the CIMOM is listening on while some clients do not. To cover both cases, the port is exposed as an optional input parameter that defaults to port 5988. This is the default Pegasus CIMOM listener port. The Java API that is provided through Sun's Wbem Services does not expose the CIMOM port.

Protocols: Most CIMOMs support either an RMI or HTTP protocol for communicating with the CIMOM. The testing that has been done shows that the HTTP protocol is not as stable, and in some cases, not fully implemented in the CIMOM. Because of this, the protocol currently defaults to RMI. The actual parameters for the WBEM Services CIMOM for the protocol are: CIMClient.CIM_RMI or CIMClient.CIM_XML.

Fetchlet Operations: The WBEM APIs are very flexible at allowing clients to traverse the class hierarchies that are defined and their associations. At this point in time, the options on accessing CIM data from an EMD are restricted to counting, getting the properties of classes, and CIMOM status. These are the more important operations that need to be performed for monitoring. As additional requirements come in, we can

add new operations to support them if necessary. For the prototype, only the count operation has been implemented.

Authentication: Most CIMOMs provide APIs to support authentication through a user identity mechanism. The majority of the CIMOMs have not implemented the API, so this capability is really a no-op. In any case, we've supplied the capability in the fetchlet so that as CIMOM implementations catch up with the standard, we'll have the necessary support in place.

**Examples**

The Wbem fetchlet supports three basic operations. At this point, the fetchlet only handles one operation at a time, so you cannot mix count, status, and value operations within a single fetchlet call. Example 1 shows how to write the metadata for a COUNT operation:

**Example 1: COUNT Operation Metadata**

```
<Metric NAME="Load" TYPE="TABLE">
    <Display>
       <Label NLSID="wbem_cimom_load">Load</Label>
     </Display>
     <TableDescriptor>
       <ColumnDescriptor NAME="Active Clients" TYPE="NUMBER" IS_KEY="FALSE">
           <Display>
              <Label NLSID="wbem_cimom_active_clients">Active CIMOM Clients</
Label>
           </Display>
        </ColumnDescriptor>
     </TableDescriptor>
    <QueryDescriptor FETCHLET_ID="Wbem">
        <Property NAME="username" SCOPE="GLOBAL">guest</Property>
        <Property NAME="password" SCOPE="GLOBAL">guest</Property>
        <Property NAME="CIMClassname" SCOPE="GLOBAL">EX_SFLProvider</Property>
        <Property NAME="operation" SCOPE="GLOBAL">COUNT</Property>
    </QueryDescriptor>
  </Metric>
```

The FETCHLET_ID is identified as Wbem. Property names are passed to the fetchlet for the required parameters user name, password, and CIMClassname. The operation is identified as COUNT.

The following example shows how to implement a Response Status metric to determine whether the CIMOM is running or not. It returns a value of 1 if the connection to the CIMOM is successful, otherwise 0.

**Example 2: Response Status Metric**

```
  <Metric NAME="Response" TYPE="TABLE">
     <Display>
       <Label NLSID="wbem_cimon_response">Response</Label>
     </Display>
     <TableDescriptor>
       <ColumnDescriptor NAME="Status" TYPE="NUMBER" IS_KEY="FALSE">
           <Display>
              <Label NLSID="wbem_cimom_response_status">Status</Label>
           </Display>
        </ColumnDescriptor>
     </TableDescriptor>
    <QueryDescriptor FETCHLET_ID="Wbem">
        <Property NAME="username" SCOPE="GLOBAL">guest</Property>
```

```
            <Property NAME="password" SCOPE="GLOBAL">guest</Property>
            <Property NAME="operation" SCOPE="GLOBAL">STATUS</Property>
        </QueryDescriptor>
    </Metric>
```

The default operation is the VALUES operation. It is used to fetch the values of a class that is defined in the CIMOM.

In the final example, the EX_Teacher class is accessed and fetches the name column. Name is the key of the class and of the new metric being defined, so the IS_KEY property is set to true. The CIM class properties will be mapped to the Enterprise Manager columns in the order that they are specified in the properties property. In this case, there is only 1 property - Name.

**Example 3: Single Property Fetched for a Class**

```
<Metric NAME="EX_Teacher" TYPE="TABLE">
    <Display>
      <Label NLSID="wbem_EX_Teacher">EX_Teacher Class</Label>
    </Display>
    <TableDescriptor>
      <ColumnDescriptor NAME="Name" TYPE="STRING" IS_KEY="TRUE">
          <Display>
            <Label NLSID="wbem_ex_teacher_name">Name</Label>
          </Display>
        </ColumnDescriptor>
    </TableDescriptor>
    <QueryDescriptor FETCHLET_ID="Wbem">
      <Property NAME="username" SCOPE="GLOBAL">guest</Property>
      <Property NAME="password" SCOPE="GLOBAL">guest</Property>
      <Property NAME="CIMClassname" SCOPE="GLOBAL">EX_Teacher</Property>
      <Property NAME="properties" SCOPE="GLOBAL">Name</Property>
    </QueryDescriptor>
</Metric>
```

If multiple properties are fetched for a class, semi-colons should separate them. The properties should be provided in the order that the column descriptors are specified for the metric table definition.

**Example 4: Multiple Properties Fetched for a Class**

```
<Metric NAME="EX_SFLProvider" TYPE="TABLE">
    <Display>
      <Label NLSID="wbem_EX_SFLProvider">EX_SFLProvider Class</Label>
    </Display>
    <TableDescriptor>
      <ColumnDescriptor NAME="Name" TYPE="STRING" IS_KEY="TRUE">
          <Display>
            <Label NLSID="wbem_ex_sfl_name">Name</Label>
          </Display>
        </ColumnDescriptor>
      <ColumnDescriptor NAME="Win" TYPE="NUMBER" IS_KEY="FALSE">
          <Display>
            <Label NLSID="wbem_ex_sfl_win">Win</Label>
          </Display>
        </ColumnDescriptor>
      <ColumnDescriptor NAME="Lost" TYPE="NUMBER" IS_KEY="FALSE">
          <Display>
            <Label NLSID="wbem_ex_sfl_lost">Lost</Label>
          </Display>
        </ColumnDescriptor>
```

```
            </TableDescriptor>
            <QueryDescriptor FETCHLET_ID="Wbem">
              <Property NAME="username" SCOPE="GLOBAL">guest</Property>
              <Property NAME="password" SCOPE="GLOBAL">guest</Property>
              <Property NAME="CIMClassname" SCOPE="GLOBAL">EX_SFLProvider</Property>
              <Property NAME="properties" SCOPE="GLOBAL">Name;Win;Lost</Property>
            </QueryDescriptor>
          </Metric>
```

# JMX Fetchlet

The JMX fetchlet retrieves Java Management Extensions (JMX) attributes (or invokes a JMX operation) from an MBean and returns the result as a (table) metric. If the ObjectName specified is an ObjectName pattern, then multiple rows are returned. Each row corresponds to an MBean matching the specified ObjectName pattern.

**Input Parameters**

**Table 20-17    JMX Fetchlet Major Input Parameters**

| Name | Type | Description | Use |
|------|------|-------------|-----|
| MachineName | String | MBean server host name | Optional |
| Port | | Port on which the MBean server is listening for new connections | Optional |
| UserName | String | User name for JMX connections, if required | Required |
| password | String | Password for JMX connections, if required | Required |
| protocol | String | Protocol used for the connection | Optional |
| service | String | Service used for connection | Optional |
| serviceURL | String | serviceURL used for JMX connection. This is instead of the previous MachineName, Port, protocol, and service properties.<br>**Note**: For middleware targets, the serviceURL can be obtained from either the farm or managedServer association depending on whether metric needs to be collected from AdminServer or the managed server. | Required (unless MachineName and Port are specified) |
| Metric | String | Mbean object name (or if MetricService=true, the DMS table name) | Required |
| columnOrder | String | Semi colon separated list of JMX attributes for the previous MBean corresponding to the column definitions in the TableDescriptor of the metric. | Required |
| operation | String | Name of the JMX operation to be invoked.In this case, the columnOrder represents the values from the return object to be populated in the Metric. (Oracle recommends using jmxcli to generate this). | Optional |

**Table 20-17    (Cont.) JMX Fetchlet Major Input Parameters**

| Name | Type | Description | Use |
|------|------|-------------|-----|
| arguments | String | The XML representing the arguments for the JMX operation. Oracle recommends using jmxcli to generate this. | Optional |
| MetricService | Boolean | MetricService=true implies that the metric is retrieved by the Oracle-specific DMS Metric Service. In this case the previous columnOrder property is a list of column names and the 'metric' property indicates the actual DMS table name. | Optional |
| identityCol | String | The Mbean object name key (or a semi-colon separated list of keys) that will be extracted from the Mbean ObjectName and surfaced as key columns in the resultant metric.If the value 'canonical' is specified, an additional key metric column with the complete Mbean object name is returned by the fetchlet.This property makes sense only if the previous metric property is an ObjectName pattern that matches more than one Mbean on the server. | Optional |
| autoRowID | String | Prefix for an automatically generated key column. The suffix is sequential numbers starting at 1.For example, autoRowID set to ROW_ generates a key column at position 0 with values ROW_1, ROW_2, and so on up to the number of rows returned.This is usually the case if none of the other columns (JMX attributes selected) are unique and multiple rows are returned as a result of multiple mbeans matches and mbean pattern. | Optional |
| useCache | Boolean | Applicable only when MetricService=true and indicates if metric service cache needs to be used | Optional |
| ServerNames | String | Applicable only when MetricService=true and is a semicolon list of server names from which the DMS metrics need to be retrieved. This is relevant only when collecting these metrics from the AdminServer (that is, serviceURL points to AdminServer through farm association), which has metrics from all managed servers | Optional |

**Table 20-17    (Cont.) JMX Fetchlet Major Input Parameters**

| Name | Type | Description | Use |
|---|---|---|---|
| valueWhenNoMBean | Number | Typically used for response metrics and has the value that the fetchlet returns as a single row and column when no mbeans are found that match the given mbean pattern (in the previous metric property). | Optional |
| valueWhenDown | Number | Typically used for response metrics. This has the value that the fetchlet returns as a single row and column when the connection to the server fails due to a connection exception (indicating that the server is down). | Optional |
| admlMap | String | Applicable only when MetricService=true and is an XML snippet that indicates what adml parameters need to be passed for this adml table.(Oracle recommends using jmxcli to generate this). | Optional |

**Notes**:

1. The JMX fetchlet is used to retrieve primarily JMX attributes from Mbeans on a target MbeanServer. It can also retrieve attributes from multiple MBeans of the same kind in the form of a table (with multiple rows where each row represents a matching MBean).

   For example, if an MBean ObjectName pattern specifies servlets, (that is, `*:Type=ServletRuntime,*` in the metric property), and the columnOrder specifies A1;A2;A3, then the resultant metric will have one row for each servlet.

2. If the metric data must be obtained using a JMX operation (this is not typical for collecting metrics), then the QueryDescriptor property operation must specify the JMX operation name and the arguments are an XML representation of the parameters to be passed into the JMX operation.

   For example, the following QueryDescriptor indicates the invocation of a JMX operation called "getNumUserSessions" with a single string argument with a value="total".

   **Example: Specifying a JMX Operation Name**

```
<Metric NAME="GetNumUserSessions" TYPE="TABLE" USAGE_TYPE="HIDDEN">
     <Display>
        <Label NLSID="GetNumUserSession">GetNumUserSession</Label>
     </Display>
     <TableDescriptor>
        <ColumnDescriptor NAME="Get Num User  Sessions" TYPE="STRING">
           <Display>
              <Label NLSID="Get Num User  Sessions">Get NumUser  Sessions</
Label>
           </Display>
        </ColumnDescriptor>
     </TableDescriptor>
     <QueryDescriptor FETCHLET_ID="JMX">
       <Property NAME="serviceURL" SCOPE="ASSOCTGT"
```

```
ASSOCIATION_NAME="farm">serviceURL</Property>
            <Property NAME="UserName" SCOPE="ASSOCTGT" OPTIONAL="TRUE"
ASSOCIATION_NAME="farm">UserName</Property>
            <Property NAME="password" SCOPE="ASSOCTGT" OPTIONAL="TRUE"
ASSOCIATION_NAME="farm">password</Property>
             <Property NAME="instName.parameter" SCOPE="INSTANCE">instName</
Property>
            <Property NAME="metric"
SCOPE="GLOBAL">oracle.forms.FormsJ2EEapplication.%instName.parameter%:,type=R
untime,*</Property>
            <Property NAME="operation" SCOPE="GLOBAL">getNumUserSessions</
Property>
            <Property NAME="columnOrder" SCOPE="GLOBAL">getNumUserSessions</
Property>
            <Property NAME="arguments" SCOPE="GLOBAL">            <!
[CDATA[<arguments>
    <argument type="java.lang.String">
       <value>total</value>
    </argument>
</arguments>]]>
        </Property>
     </QueryDescriptor>
    </Metric>
```

**3.** A QueryDescriptor for the JMX fetchlet contains JMX connection information. This is usually in the form of a serviceURL. If the serviceURL property is not available in the QueryDescriptor, then the combination of MachineName, Port, protocol, and service properties must be present in the QueryDescriptor to provide connection information to the JMX fetchlet.

# Web Service Fetchlet

In target metadata files generated by the Web Services Command-Line tool, the `<QueryDescriptor>` element specifies the properties that will be passed to the Web Services fetchlet when being invoked.

> **✎ Note:**
>
> From Release 13.1 onwards, the Web Service fetchlet is available from the Enterprise Manager for Fusion Apps and the Enterprise Manager for Fusion Middleware plug-ins.
>
> To use this fetchlet, you must have the Enterprise Manager for Fusion Apps and the Enterprise Manager for Fusion Middleware plug-ins deployed on both the OMS and the Management Agent.

**Input Parameters**

Table 20-18 lists the supported properties:

**Table 20-18    Web Service Fetchlet Properties**

| Name | Description | Use | Comments |
| --- | --- | --- | --- |
| ServiceName | Web service name | Required.<br>Service Name must be prefixed with a valid namespace. | All referenced namespaces are specified by the property *"Namespace"* |
| PortName | Web service port name | Required.<br>Port Name must be prefixed with a valid namespace. | All referenced namespaces are specified via the property *"Namespace"* |
| OperationName | Web service operation name | Required.<br>Operation Name must be prefixed with a valid namespace. | All referenced namespaces are specified by the property *"Namespace"* |
| ServiceEndpoint | Web service endpoint | Required.<br>A valid URL. | |
| WsdlURL | Web service WSDL URL | Optional.<br>A valid URL. | Required only if it is a RPC/Encoded Web service |
| ParameterStyle | SOAP parameter mapping style | Optional.<br>- *BARE*<br>- *WRAPPED* | Optional only if it is a RPC/Encoded or REST-ful Web service |
| Payload | Web service operation request payload | Required.<br>Must be specified using the CDATA section. | |
| SOAPBindingStyle | SOAP binding style | Optional.<br>- *DOCUMENT*<br>- *RPC* | Optional only if it is a RPC or Encoded Web service |
| SOAPBindingUse | SOAP binding use | Optional<br>- *ENCODED*<br>- *LITERAL* | Optional only if it is a RPC or Encoded Web service |
| SOAPVersion | SOAP version | Optional<br>- *SOAP_1_1*<br>- *SOAP_1_2* | Optional only if it is an RPC or Encoded Web service |
| MessageType | Web service message type | Optional<br>- *SOAP*<br>- *REST* | Optional only if it is a RPC or Encoded Web service |
| SecurityPolicy | Security policy | Required<br>- *NONE*<br>- *BASIC_AUTHENTICATION* | |

**Table 20-18    (Cont.) Web Service Fetchlet Properties**

| Name | Description | Use | Comments |
|------|-------------|-----|----------|
| Namespace | Set of all namespaces referenced | Optional. Contains all the namespaces referenced in the metric<br><br>Specify using notation: [ns0="uri0"][ns1="uri1"]<br><br>Example: [ns0="http://type.abc.com"]<br><br>[ns1="http://app.abc.com"] | |
| ColType | Collection result column type | Required<br><br>List of metric column type (separated by comma)<br><br>Example: msgId:STRING,source:STRING,detail:STRING | |
| RowType | Collection result row type | Required<br><br>List of XPath expression corresponding to metric columns (separated by comma)<br><br>For example: //ns0:eventResponse/msgId,<br><br>//ns0:eventResponse/source, | |
| SSLKeyStoreCredential | SSL keystore credentialSet name | Optional<br><br>A valid CredentialSet of a Store Credential Type defined in the <CredentialInfo> | Must be defined as a monitoring credential. |
| SSLTrustStoreCredential | SSL truststore credentialset name | Optional<br><br>A valid CredentialSet of a StoreCredential Type defined in the <CredentialInfo> | Must be defined as a monitoring credential. |
| UserCredential | User token credentialset name | Optional<br><br>A valid CredentialSet of a AliasCredential or CSFKeyCrdential Type defined in the <CredentialInfo> | Must be defined as a monitoring credential. |
| ValueWhenDown | Default response when target is down | Required (only for response metric). Not required for regular metric. For Response metric, when a target is down, this value (if specified) will be returned. | A target is considered as down when the Fetchlet catches a ConnectionException. |

**Examples**

Example 1 provides an example of a metric definition for Remote Procedure Call (RPC) or encoded Web services and Example 2 provides an example of a metric definition for doc or literal Web services.

**Example 1: Metric Definition for RPC or Encoded Web Service**

```
<Metric NAME="getVacantRooms" TYPE="TABLE">
  <Display>
    <Label NLSID="NLSID_GET_VACANT_ROOMS">getVacantRooms</Label>
  </Display>

  <TableDescriptor>
    <ColumnDescriptor IS_KEY="TRUE" NAME="roomID" TYPE="STRING">
      <Display>
        <Label NLSID="COL_ROOM_ID">roomID</Label>
      </Display>
    </ColumnDescriptor>
    <ColumnDescriptor IS_KEY="FALSE" NAME="floor" TYPE="STRING">
      <Display>
        <Label NLSID="COL_FLOOR">floor</Label>
      </Display>
    </ColumnDescriptor>
    <ColumnDescriptor IS_KEY="FALSE" NAME="number" TYPE="STRING">
      <Display>
        <Label NLSID="COL_NUMBER">number</Label>
      </Display>
    </ColumnDescriptor>
    <ColumnDescriptor IS_KEY="FALSE" NAME="rate" TYPE="STRING">
      <Display>
        <Label NLSID="COL_RATE">rate</Label>
      </Display>
    </ColumnDescriptor>
    <ColumnDescriptor IS_KEY="FALSE" NAME="roomType" TYPE="STRING">
      <Display>
        <Label NLSID="COL_ROOM_TYPE">roomType</Label>
      </Display>
    </ColumnDescriptor>
    <ColumnDescriptor IS_KEY="FALSE" NAME="smoking" TYPE="STRING">
      <Display>
        <Label NLSID="COL_SMOKING">smoking</Label>
      </Display>
    </ColumnDescriptor>
    <ColumnDescriptor IS_KEY="FALSE" NAME="available" TYPE="STRING">
      <Display>
        <Label NLSID="COL_AVAILABLE">available</Label>
      </Display>
    </ColumnDescriptor>
  </TableDescriptor>
  <QueryDescriptor FETCHLET_ID="OWSM_WSF">
   <Property NAME="SecurityPolicy" SCOPE="INSTANCE">NONE</Property>
    <Property NAME="WsdlURL" SCOPE="INSTANCE">wsdlURL</Property>
    <Property NAME="ServiceEndpoint" SCOPE="INSTANCE">serviceURL</Property>
    <Property NAME="ServiceName" SCOPE="GLOBAL">ns0:SimpleHotelServiceRE</
Property>
    <Property NAME="PortName" SCOPE="GLOBAL">ns0:HotelService</Property>
    <Property NAME="OperationName" SCOPE="GLOBAL">getVacantRooms</Property>
    <Property NAME="MessageType" SCOPE="GLOBAL">SOAP</Property>
    <Property NAME="SOAPBindingStyle" SCOPE="GLOBAL">RPC</Property>
    <Property NAME="SOAPBindingUse" SCOPE="GLOBAL">ENCODED</Property>
    <Property NAME="ParameterStyle" SCOPE="GLOBAL">BARE</Property>
    <Property NAME="SOAPVersion" SCOPE="GLOBAL">SOAP_1_1</Property>
    <Property NAME="Namespace" SCOPE="GLOBAL"><![CDATA[[ns1="http://
hotel.apps.muws/"][ns0="http://hotel.apps.muws/rpc/"]]></Property>
    <Property NAME="RowType" SCOPE="GLOBAL">//ns1:getVacantRoomsResponse/
return/item/@roomID,//ns1:getVacantRoomsResponse/return/item/floor,
```

```
     //ns1:getVacantRoomsResponse/return/item/number,//
ns1:getVacantRoomsResponse/return/item/rate,//ns1:getVacantRoomsResponse/return/
item/roomType,
     //ns1:getVacantRoomsResponse/return/item/smoking,//
ns1:getVacantRoomsResponse/return/item/available</Property>
     <Property NAME="ColType"
SCOPE="GLOBAL">roomID:STRING,floor:STRING,number:STRING,rate:STRING,roomType:STRI
NG,smoking:STRING,available:STRING</Property>
     <Property NAME="Payload" SCOPE="GLOBAL"><![CDATA[<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:ns="http://hotel.apps.muws/"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
          <soap:Body soap:encodingStyle="">
            <ns:getVacantRooms/>
          </soap:Body>
        </soap:Envelope>]]></Property>
   </QueryDescriptor>
  </Metric>
```

**Example 2: Metric Definition for Doc or Literal Web Service**

```
    <Metric NAME="square" TYPE="TABLE">
      <Display>
        <Label NLSID="NLSID_SQUARE">square</Label>
      </Display>
      <TableDescriptor>
        <ColumnDescriptor IS_KEY="FALSE" NAME="arg0" TYPE="STRING">
          <Display>
            <Label NLSID="COL_ARG0">arg0</Label>
          </Display>
        </ColumnDescriptor>
      </TableDescriptor>
      <QueryDescriptor FETCHLET_ID="OWSM_WSF">
        <Property NAME="SecurityPolicy" SCOPE="INSTANCE">NONE</Property>
        <Property NAME="ServiceEndpoint" SCOPE="INSTANCE">serviceURL</Property>
        <Property NAME="ServiceName" SCOPE="GLOBAL">ns0:CalculatorService</
Property>
        <Property NAME="PortName" SCOPE="GLOBAL">ns0:CalculatorPort</Property>
        <Property NAME="OperationName" SCOPE="GLOBAL">square</Property>
        <Property NAME="MessageType" SCOPE="GLOBAL">SOAP</Property>
        <Property NAME="SOAPBindingStyle" SCOPE="GLOBAL">DOCUMENT</Property>
        <Property NAME="SOAPBindingUse" SCOPE="GLOBAL">LITERAL</Property>
        <Property NAME="ParameterStyle" SCOPE="GLOBAL">WRAPPED</Property>
        <Property NAME="SOAPVersion" SCOPE="GLOBAL">SOAP_1_1</Property>
        <Property NAME="Namespace" SCOPE="GLOBAL"><![CDATA[[ns0="http://
tests.jaxws.oracle.com/"]
[ns1="http://www.oracle.com/jaxws/tests"]]]></Property>
        <Property NAME="RowType" SCOPE="GLOBAL">//ns1:squareResponse/arg0</
Property>
        <Property NAME="ColType" SCOPE="GLOBAL">arg0:STRING</Property>
        <Property NAME="Payload" SCOPE="GLOBAL"><![CDATA[<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
            <soap:Body xmlns:ns1="http://www.oracle.com/jaxws/tests">
              <ns1:square>
                <arg0>%square.arg00001%</arg0>
              </ns1:square>
            </soap:Body>
          </soap:Envelope>]]></Property>
```

```
        </QueryDescriptor>
      </Metric>
```

# Using Credentials for Authentication

If basic authentication is required, then you must configure or define the following in the metric definition:

1. Set the SecurityPolicy property to BASIC_AUTHENTICATION:

```
    <Property NAME="SecurityPolicy" SCOPE="INSTANCE">BASIC_AUTHENTICATION</
Property>
```

2. Add the following properties to the `<QueryDescriptor>` element:

```
     <Property NAME="UserCredential" SCOPE="GLOBAL"> UserCredentialSet </
Property>
      <CredentialRef NAME="UserCredentialSet">UserCredentialSet</CredentialRef>
```

3. Define the credential type after the `<Metric>` tag:

```
.........
        <Property NAME="UserCredential" SCOPE="GLOBAL">UserCredentialSet     </
Property>
          <CredentialRef NAME="UserCredentialSet">UserCredentialSet </
CredentialRef>
        </QueryDescriptor>
      </Metric>
          <CredentialInfo>
      <CredentialType NAME="AliasCredential">
        <Display>
          <Label NLSID="CRED_TYPE">Alias Credential Type</Label>
        </Display>
        <CredentialTypeColumn NAME="Alias">
          <Display>
            <Label NLSID="CRED_ALIAS">Alias (i.e. username, encryption key,
signature key, etc)</Label>
          </Display>
        </CredentialTypeColumn>
        <CredentialTypeColumn NAME="Password">
          <Display>
            <Label NLSID="CRED_PASSWORD">Password for the alias</Label>
          </Display>
        </CredentialTypeColumn>
      </CredentialType>
      <CredentialSet NAME="UserCredentialSet" USAGE="MONITORING">
        <AllowedCredType TYPE="AliasCredential"/>
      </CredentialSet>
    </CredentialInfo>
```

**Example: Using Keystore and Truststore for SSL**

```
............
        <Property NAME="SSLTrustStoreCredential"
SCOPE="GLOBAL">SSLTrustStoreCredentialSet</Property>
        <Property NAME="SSLKeyStoreCredential"
SCOPE="GLOBAL">SSLKeyStoreCredentialSet</Property>
NAME="SSLTrustStoreCredentialSet">SSLTrustStoreCredentialSet</CredentialRef>
        <CredentialRef NAME="SSLKeyStoreCredentialSet">SSLKeyStoreCredentialSet</
CredentialRef>
      </QueryDescriptor>
    </Metric>
```

```
<CredentialInfo>
  <CredentialType NAME="StoreCredential">
    <Display>
      <Label NLSID="CRED_TYPE">Store Credential Type</Label>
    </Display>
    <CredentialTypeColumn NAME="StoreLocation">
      <Display>
        <Label NLSID="CRED_STORE_LOCATION">Store Location</Label>
      </Display>
    </CredentialTypeColumn>
    <CredentialTypeColumn NAME="StoreType">
      <Display>
        <Label NLSID="CRED_STORE_TYPE">Store Type</Label>
      </Display>
    </CredentialTypeColumn>
    <CredentialTypeColumn NAME="StorePassword">
      <Display>
        <Label NLSID="CRED_STORE_PASSWORD">Store Password</Label>
      </Display>
    </CredentialTypeColumn>
  </CredentialType>
  <CredentialSet NAME="SSLTrustStoreCredentialSet" USAGE="MONITORING">
    <AllowedCredType TYPE="StoreCredential"/>
  </CredentialSet>
  <CredentialSet NAME="SSLKeyStoreCredentialSet" USAGE="MONITORING">
    <AllowedCredType TYPE="StoreCredential"/>
  </CredentialSet>
</CredentialInfo>
```

# WS-Management Fetchlet

In target metadata files generated by the *wsmancli* Command-Line Tool, the `<QueryDescriptor>` element specifies the properties that will be passed to the WSManagement fetchlet when being invoked.

> **Note:**
>
> From Release 13.1 onwards, the WS-Management fetchlet is available from the Enterprise Manager for Fusion Middleware plug-in.
>
> To use this fetchlet, you must have the Enterprise Manager for Fusion Middleware plug-in deployed on both the OMS and the Management Agent.

**Input Parameters**

Table 20-19 provides a complete list of the supported properties:

**Table 20-19    WS Management Fetchlet Properties**

| Name | Description | Use |
| --- | --- | --- |
| ResourceURI | URI of a resource class representation or instance representation (wsman:ResourceURL) | Required<br>Any valid URI according to RFC 3986 |

**Table 20-19    (Cont.) WS Management Fetchlet Properties**

| Name | Description | Use |
|---|---|---|
| To | Transport address of a service (wsa:To). | Required<br><br>Any valid network transport address. |
| Action | wsa:Action identifies which operation is to be carried out against the resource. | Required<br><br>Current release only supports "http://schemas.xmlsoap.org/ws/2004/09/transfer/Get". |
| TransferOperation | Name of the WS-Transfer operation. | Required<br><br>Current release only supports "GET". |
| Locale | Specifies the language that the client requests (and sometimes requires) and the response text to be translated into (wsman:Locale) | Optional<br><br>Any valid value for the standard XML attribute xml:lang |
| MaxEnvelopeSize | The size to indicate that client expects a response whose total SOAP envelope does not exceed the specified number of octets (wsman:MaxEnvelopeSize) | Optional<br><br>Value should not be less than 8192 |
| OperationTimeout | The value to indicate that client expects a response or a fault within the specified time (wsman:OperationTimeout). | Optional<br><br>Specify the value using format xs:duration (see http://www.w3.org/2001/XMLSchema:duration). |
| OptionSet | A set of switches to the service to modify or refine the nature of the request (wsman:OptionSet). | Optional<br><br>Specify the values using the notation:<br><br>[<OptionName1>, value:<value1>, type:<type1>, mustComply:<true\|false>][<OptionName2>, value:<value2>, type:<type>, mustComply:<true\|false>][...] |
| ReplyTo | The header to be present in all request messages when a reply is required (wsa:ReplyTo). | Optional<br><br>It should be either a valid address for a new connection using any transport supported by the service or the URI http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous (see WS-Addressing) |
| SelectorSet | Set of selectors that identify the instance of resource to be accessed (wsman:SelectorSet) | Required<br><br>Specify the value using the format below:<br><br>*[S1, V1][S2, V2]...[Sn, Vn]*<br>Where<br>- S1, S2, ..., Sn are Selector names<br>- V1, V2, ..., Vn are Selector values |

ORACLE

**Table 20-19    (Cont.) WS Management Fetchlet Properties**

| Name | Description | Use |
| --- | --- | --- |
| SecurityPolicy | Security policy | Required<br>- *NONE*<br>- *BASIC_AUTHENTICATION* |
| Namespace | Set of all namespaces referenced | Required<br>Specify using notation: [ns0="uri0"] [ns1="uri1"]..<br>Example: [ns0="http://type.abc.com"] [ns1="http://app.abc.com"] |
| ColType | Collection result column type | Required<br>List of metric column types (separated by comma)<br>Example: msgId:STRING,source:STRING,detail: STRING |
| RowType | Collection result row type | Required<br>List of XPath expression corresponding to metric columns (separated by comma)<br>For example: //ns0:eventResponse/ msgId,//ns0:eventResponse/source |
| SSLKeyStoreCredential | SSL keystore credentialSet name | Optional<br>A valid CredentialSet of a Store Credential Type defined in the <CredentialInfo> |
| SSLTrustStoreCredential | SSL truststore credentialSet name | Optional<br>A valid CredentialSet of a StoreCredential Type defined in the <CredentialInfo> tag. |
| UserCredential | User token credentialSet name | Optional<br>A valid CredentialSet of a AliasCredential or CSFKeyCrdential Type defined in the <CredentialInfo> tag. |
| ValueWhenDown | Default response when target is down | Required (only for response metric). Not required for regular metric. For Response metric, when a target is down, this value (if specified) will be returned.<br>A target is considered as down when the Fetchlet catches a ConnectionException |

**Example: Metric definition for using the WS-Management Fetchlet**

The following example provides an example of a metric definition using the WS-Management fetchlet.

```
<Metric NAME="trafficLight" TYPE="TABLE">
  <Display>
```

```
                    <Label NLSID="NLSID_TRAFFIC_LIGHT">trafficLight</Label>
                  </Display>
                  <TableDescriptor>
                    <ColumnDescriptor IS_KEY="YES" NAME="name" TYPE="STRING">
                      <Display>
                        <Label NLSID="COL_NAME">name</Label>
                      </Display>
                    </ColumnDescriptor>
                    <ColumnDescriptor IS_KEY="FALSE" NAME="color" TYPE="STRING">
                      <Display>
                        <Label NLSID="COL_COLOR">color</Label>
                      </Display>
                    </ColumnDescriptor>
                    <ColumnDescriptor IS_KEY="FALSE" NAME="x" TYPE="STRING">
                      <Display>
                        <Label NLSID="COL_X">x</Label>
                      </Display>
                    </ColumnDescriptor>
                    <ColumnDescriptor IS_KEY="FALSE" NAME="y" TYPE="STRING">
                      <Display>
                        <Label NLSID="COL_Y">y</Label>
                      </Display>
                    </ColumnDescriptor>
                  </TableDescriptor>
                  <QueryDescriptor FETCHLET_ID="WSManagementFetchlet">
                    <Property NAME="SecurityPolicy" SCOPE="INSTANCE">NONE</Property>
                    <Property NAME="ResourceURL" SCOPE="INSTANCE">resourceURL</Property>
                    <Property NAME="To" SCOPE="INSTANCE">To address</Property>
                    <Property NAME="OptionSet" SCOPE="INSTANCE">optionSet</Property>
                    <Property NAME="Locale" SCOPE="INSTANCE">locale</Property>
                    <Property NAME="MaxEnvelopeSize" SCOPE="INSTANCE">maxEnvelopeSize</
Property>
                    <Property NAME="OperationTimeout" SCOPE="INSTANCE">operationTimeout</
Property>
                    <Property NAME="Namespace" SCOPE="GLOBAL"> <![CDATA[[ns1="http://
schemas.wiseman.dev.java.net/traffic/1/light.xsd"]
                        [ns0="http://www.w3.org/2001/XMLSchema"]
                        [wsa="http://www.w3.org/2005/08/addressing"]
                        [env="http://www.w3.org/2003/05/soap-envelope"]]]></Property>
                    <Property NAME="RowType" SCOPE="GLOBAL">//ns1:trafficlight/ns1:name,//
ns1:trafficlight/ns1:color,//ns1:
                    <Property NAME="ColType"
SCOPE="GLOBAL">name:STRING,color:STRING,x:STRING,y:STRING</Property>
                    <Property NAME="ReplyTo" SCOPE="GLOBAL">http://www.w3.org/2005/08/
addressing/role/anonymous</Property>
                    <Property NAME="Action" SCOPE="GLOBAL">http://schemas.xmlsoap.org/ws/
2004/09/transfer/Get</Property>
                    <Property NAME="TransferOperation" SCOPE="GLOBAL">GET</Property>
                    <Property NAME="SelectorSet" SCOPE="GLOBAL">[name,Light1]</Property>
                  </QueryDescriptor>
                </Metric>
```

## Using Credentials

If basic authentication is required, then configure or define in the metric definition:

1. Set the SecurityPolicy property to BASIC_AUTHENTICATION:

```
    <Property NAME="SecurityPolicy" SCOPE="INSTANCE">BASIC_AUTHENTICATION</
    Property>
```

**2.** Add the following properties in the `<QueryDescriptor>` tag:

```
    <Property NAME="UserCredential" SCOPE="GLOBAL"> UserCredentialSet </
Property>
    <CredentialRef NAME="UserCredentialSet">UserCredentialSet</CredentialRef>
```

**3.** Define the credential type after the `<Metric>` tag:

```
        ………
    <Property NAME="UserCredential" SCOPE="GLOBAL">UserCredentialSet </
Property>
        <CredentialRef NAME="UserCredentialSet">UserCredentialSet </
CredentialRef>
      </QueryDescriptor>
    </Metric>
  <CredentialInfo>
    <CredentialType NAME="AliasCredential">
      <Display>
        <Label NLSID="CRED_TYPE">Alias Credential Type</Label>
      </Display>
      <CredentialTypeColumn NAME="Alias">
        <Display>
          <Label NLSID="CRED_ALIAS">Alias (i.e. username, encryption key,
signature key, etc)</Label>
        </Display>
      </CredentialTypeColumn>
      <CredentialTypeColumn NAME="Password">
        <Display>
          <Label NLSID="CRED_PASSWORD">Password for the alias</Label>
        </Display>
      </CredentialTypeColumn>
    </CredentialType>
    <CredentialSet NAME="UserCredentialSet" USAGE="MONITORING">
      <AllowedCredType TYPE="AliasCredential"/>
    </CredentialSet>
  </CredentialInfo>
```

**Example: Using Keystore and Truststore for SSL**

```
    …………
    <Property NAME="SSLTrustStoreCredential"
SCOPE="GLOBAL">SSLTrustStoreCredentialSet</Property>
    <Property NAME="SSLKeyStoreCredential"
SCOPE="GLOBAL">SSLKeyStoreCredentialSet</Property>
<CredentialRef NAME="SSLTrustStoreCredentialSet">SSLTrustStoreCredentialSet</
CredentialRef>
    <CredentialRef NAME="SSLKeyStoreCredentialSet">SSLKeyStoreCredentialSet</
CredentialRef>
    </QueryDescriptor>
  </Metric>
  <CredentialInfo>
    <CredentialType NAME="StoreCredential">
      <Display>
        <Label NLSID="CRED_TYPE">Store Credential Type</Label>
      </Display>
      <CredentialTypeColumn NAME="StoreLocation">
        <Display>
          <Label NLSID="CRED_STORE_LOCATION">Store Location</Label>
        </Display>
      </CredentialTypeColumn>
      <CredentialTypeColumn NAME="StoreType">
        <Display>
          <Label NLSID="CRED_STORE_TYPE">Store Type</Label>
```

```
            </Display>
          </CredentialTypeColumn>
          <CredentialTypeColumn NAME="StorePassword">
            <Display>
               <Label NLSID="CRED_STORE_PASSWORD">Store Password</Label>
            </Display>
          </CredentialTypeColumn>
        </CredentialType>
        <CredentialSet NAME="SSLTrustStoreCredentialSet" USAGE="MONITORING">
          <AllowedCredType TYPE="StoreCredential"/>
        </CredentialSet>
        <CredentialSet NAME="SSLKeyStoreCredentialSet" USAGE="MONITORING">
          <AllowedCredType TYPE="StoreCredential"/>
        </CredentialSet>
      </CredentialInfo>
```

# REST Fetchlet

The REST fetchlet provides target monitoring for RESTful web resources. Based on input properties, this fetchlet can construct a request to communicate with the managed targets using HTTP standards. It can retrieve relevant data from the response to build and return monitoring metrics.

This release supports the following RESTful web services only:

- HTTP methods
  - GET: Define a reading access of the source without any side-effects. The resource is never changed through a GET request.
  - POST: Update an existing resource or create a new resource.
  - HEAD: vCheck if a given path is serviceable.
- Media type of request or response representations
  - application/xml (both request and response)
  - application/json (both request and response)
  - text/xml (response only)
  - application/x-www-form-urlencoded (request only)
- Authentication scheme

  Supports BASIC authentication.

## Response Processing

The fetchlet relies on response data to construct monitoring metrics. Because the response media type can be application/xml, application/json, or text/xml, different mechanisms are adapted to process the response. Table 20-20 describes the different mechanisms for each response media type.

**Table 20-20    Resonse Processing Mechanism**

| Media Type | Mechanism |
|---|---|
| application/xml | XPath Query is used for processing XML data. |
| | The fetchlet property, RowType, specifies a list of XPath expressions corresponding to metric columns (separated by comma) for retrieving column data. |
| | For example: |

```
<records>
  <ns2:Record xmlns:ns2="urn:com.office.directory">
    <name>Peter</name>
    <phone>+1 (650) 555-0100</phone>
  </ns2:Record>
  <ns2:Record xmlns:ns2="urn:com.office.directory">
    <name>John</name>
    <phone>+1 (650) 555-0185</phone>
  </ns2:Record>
</records>
```

Assume the monitoring metric has two columns (name and phone). The corresponding XPath expressions are:

- //records/ns.2:Record/name
- //records/ns.2:Record/phone

The following is an example of extracted data:

```
Peter, +1 (650) 555-0100
John, +1 (650) 555-0185
```

| Media Type | Mechanism |
|---|---|
| application/json | JSONPath is used for processing JavaScript Object Notation (JSON) data. JSONPath expressions refer to a JSON structure in the same way as XPath expressions are used in an XML document. |
| | For example: |

```
{
  "Record":
  [
    {
      "name":"Peter"
      "phone":"+1 (650) 555-0100"
    },
    {
      "name":"John"
      "phone":"+1 (650) 555-0185"
    },
  ]
}
```

Assume the monitoring metric has two columns (name and phone). The corresponding JSONPath expressions are:

- $.Record.name
- $.Record.phone

The following example is an example of extracted data:

```
Peter, +1 (650) 555-0100
John, +1 (650) 555-0185
```

**Table 20-20    (Cont.) Resonse Processing Mechanism**

| Media Type | Mechanism |
|---|---|
| text/xml | Because text is a non-structural representation, there is no way to extract any specific data from it. Instead, the entire response is returned. |

**Input Parameters**

Table 20-21 provides a complete list of the supported properties.

**Table 20-21    REST Fetchlet Properties**

| Name | Description | Optional |
|---|---|---|
| BaseURI | Base URI of the RESTful web service | No |
| RequestElementPayload | Request element payload (XML/JSON) in string format. Must be specified using the CDATA section if it is XML | Yes |
| RequestMetadata | Request metadata in XML format | No |
| SecurityPolicy | Specifies authentication scheme. Either NONE or BASIC_AUTHENTICATION | No |
| Namespace | Set of all namespaces referenced in the metric. Specify using notation: [ns0="uri0"][ns1="uri1"]...<br><br>For example:<br><br>`[ns0="http://type.abc.com"][ns1="http://app.abc.com"]` | No |
| ColType | Collection result column type. List of metric column type (separated by comma).<br><br>For example:<br><br>`msgId:STRING,source:STRING,detail:STRING` | No |
| RowType | Collection result row type. List of path (XPath or JsonPath) expressions corresponding to metric columns (separated by comma).<br><br>For example:<br><br>`//ns0:eventResponse/msgId,//ns0:eventResponse/source,//ns0:eventResponse/detail` | No |
| SSLKeyStoreCredential | SSL keystore credential set name. It must be defined as a monitoring credential and contain these credential columns:<br><br>Location, Type, Password | Yes |
| SSLTrustStoreCredential | SSL truststore credentialset name. It must be defined as a monitoring credential and contain these credential columns:<br><br>Location, Type, Password | Yes |

**Table 20-21 (Cont.) REST Fetchlet Properties**

| Name | Description | Optional |
| --- | --- | --- |
| UserCredential | User token credentialset name. It must be defined as a monitoring credential and contain these credential columns:<br><br>Alias, Password | Yes |
| ProxyHost | Host name of the proxy server to make the URL connection | Yes |
| ProxyPort | Port number of the proxy server to make the URL connection | Yes |

The following example shows an example of the Fectchlet Query Descriptor from a target metadata file. For more information about the target metadata files, see Creating Target Metadata Files .

> **Note:**
>
> The fetchlet ID is RESTFetchlet.

**Example: Fetchlet Query Descriptor**

```
<QueryDescriptor FETCHLET_ID="RESTFetchlet">
  <Property NAME="SecurityPolicy" SCOPE="INSTANCE">ListAll.SecurityPolicy</
Property>
  <Property NAME="BaseURI" SCOPE="INSTANCE">ListAll.BaseURI</Property>
  <Property NAME="Namespace" SCOPE="GLOBAL">
    <![CDATA[[ns0="urn:com.office.directory"]]]></Property>
  <Property NAME="RowType" SCOPE="GLOBAL">//ns0:Record/name,//ns0:Record/title,
    //ns0:Record/phone,//ns0:Record/building,//ns0:Record/floor,
    //ns0:Record/office</Property>
  <Property NAME="ColType" SCOPE="GLOBAL">name:STRING,title:STRING,phone:STRING
    ,building:STRING,floor:STRING,office:STRING</Property>
  <Property NAME="RequestMetadata" SCOPE="GLOBAL">
          <![CDATA[<Resource path="/">
              <Resource path="lookup/list">
                  <Method elementDefined="false"
                   accept="application/xml" name="GET"/>
              </Resource>
          </Resource>
          </Property>]]>
  <Property NAME="UserCredential" SCOPE="GLOBAL">UserCredentialSet</Property>
  <CredentialRef NAME="UserCredentialSet">UserCredentialSet</CredentialRef>
</QueryDescriptor>
```

# Using HTTPS and Self-Signed Certificates

When calling an HTTPS URL with a self-signed SSL certificate from a REST fetchlet, the credential set must be specified in the target metadata file.

```
<QueryDescriptor FETCHLET_ID="RESTFetchlet">
  ......
    <Property NAME="SSLTrustStoreCredential" SCOPE="GLOBAL">
```

```
      SSLTrustStoreCredentialSet</Property>
    <CredentialRef NAME="SSLTrustStoreCredentialSet">
     SSLTrustStoreCredentialSet</CredentialRef></QueryDescriptor>
<CredentialInfo>
  <CredentialType NAME="StoreCredential">
    <Display>
      <Label NLSID="CRED_TYPE">Store Credential Type</Label>
    </Display>
    <CredentialTypeColumn NAME="StoreLocation" IS_SENSITIVE="FALSE">
      <Display>
        <Label NLSID="CRED_STORE_LOCATION">Store Location</Label>
      </Display>
    </CredentialTypeColumn>
    <CredentialTypeColumn NAME="StoreType" IS_SENSITIVE="FALSE">
      <Display>
        <Label NLSID="CRED_STORE_TYPE">Store Type</Label>
      </Display>
    </CredentialTypeColumn>
    <CredentialTypeColumn NAME="StorePassword">
      <Display>
        <Label NLSID="CRED_STORE_PASSWORD">Store Password</Label>
      </Display>
    </CredentialTypeColumn>
  </CredentialType>
  <CredentialSet NAME="SSLTrustStoreCredentialSet" USAGE="MONITORING"
CONTEXT_TYPE="TARGET">
    <AllowedCredType TYPE="StoreCredential"/>
  </CredentialSet>
</CredentialInfo>
```

In Enterprise Manager Cloud Control 12*c* (12.1.0.3), a new fetchlet property, "SSLTrustServerCert", was added. If set to "TRUE", the fetchlet uses the non-validating mode for the server certificate, and there is no need to provide or specify the SSL trust store.

```
<Property NAME="SSLTrustServerCert" SCOPE="GLOBAL">TRUE</Property>
```

# Using REST CLI to Generate Metadata

REST CLI is a client command line tool for generating target metadata and default collection files to enable the Management Agent to monitor RESTful web resources through invoking the REST fetchlet.

Use the following `emctl` command to invoke REST CLI:

```
emctl restcli
```

Table 20-22 provides a list of the command-line arguments that you can use with the `emctl restcli` command.

**Table 20-22    Command-line Arguments Supported by REST CLI**

| Argument | Description | Example |
|---|---|---|
| metadata | Generate target metadata | -metadata |
| wadl | WADL location | -wadl=http://...... |
|  |  | -wadl=file:///..... |

**Table 20-22    (Cont.) Command-line Arguments Supported by REST CLI**

| Argument | Description | Example |
|---|---|---|
| wsdl | WSDL location | -wsdl=http://...... <br> -wsdl=file:///..... |
| username | User name to log in to the host | -username=admin |
| proxyhost | Host name of the proxy server | -proxyhost=proxy.*example*.com |
| proxyport | Port number of the proxy server | -proxyport=80 |

To use REST CLI:

1. Run the REST CLI command with the Web Application Description Language (WADL) location. For example:

   ```
   emctl restcli -wadl=http://host.us.example.com:17382/OfficeDirectoryBA/
   application.wadl
   ```

   If the WADL location is access protected, then enter a user name and password.

   The Running REST CLI example below provides an example of a user running the REST CLI tool.

2. REST CLI prompts you to enter the target type and location where the output directory will contain the generated target and collection metadata files.

3. REST CLI lists out all the available resources paths for monitoring. You must select a resource path and one of its methods to define monitoring metric for that resource.

4. REST CLI also prompts you to define the collection schedule.

   When all the information is gathered from the user, the tool generates the target and default collection metadata files under the specified output directory similar to the metadata provided in the REST CLI-Generated Target Metadata example at the end of this section.

   **Example: Running REST CLI**

   ```
   Generate Metric Metadata for REST Web Resource Monitoring

   Enter password for "weblogic" :

   Reading WADL Document at
   http://host.us.example.com:17382/OfficeDirectoryBA/application.wadl...done.


   ==> Enter the name for this target type : OfficeDirectory

   ==> Enter metadata file name [/scratch/work/metadata/OfficeDirectory.xml] :

   All resource paths available for monitoring :
   [1]    /add
   [2]    /lookup/list
   [3]    /lookup/phone
   [4]    /lookup/building/people
   [5]    /db/count
   ```

```
==> Enter the index [1-5] to select : 2
* Selected Resource Path : /lookup/list

All methods available from the selected path :
[1]   application/xml[Record] GET()
[2]   application/json[Record] GET()

==> Enter the index [1-2] to select : 1
* Selected Resource Method: application/xml[Record] GET()

Define new metric group :
==> Enter the name for this metric group [GET] : ListGet_XML

Return value(s) for the selected method :
[1]   //ns0:Record/name <string>
[2]   //ns0:Record/title <string>
[3]   //ns0:Record/phone <string>
[4]   //ns0:Record/building <string>
[5]   //ns0:Record/floor <string>
[6]   //ns0:Record/office <string>

==> Enter the index [1-6] of metric to display : 1
==> Enter the name for this metric [name] :
==> Enter the label for this metric [name] :
==> Is this a key metric <y/n>? [n] : y
==> Do you want to add another metric <y/n>? [y] :

Return value(s) for the selected method :
[1]   //ns0:Record/title <string>
[2]   //ns0:Record/phone <string>
[3]   //ns0:Record/building <string>
[4]   //ns0:Record/floor <string>
[5]   //ns0:Record/office <string>

==> Enter the index [1-5] of metric to display : 1
==> Enter the name for this metric [title] :
==> Enter the label for this metric [title] :
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for it <y/n>? [n] :
==> Do you want to add another metric <y/n>? [y] :

Return value(s) for the selected method :
[1]   //ns0:Record/phone <string>
[2]   //ns0:Record/building <string>
[3]   //ns0:Record/floor <string>
[4]   //ns0:Record/office <string>

==> Enter the index [1-4] of metric to display : 1
==> Enter the name for this metric [phone] :
==> Enter the label for this metric [phone] :
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for it <y/n>? [n] :
==> Do you want to add another metric <y/n>? [y] :

Return value(s) for the selected method :
[1]   //ns0:Record/building <string>
[2]   //ns0:Record/floor <string>
[3]   //ns0:Record/office <string>

==> Enter the index [1-3] of metric to display : 1
==> Enter the name for this metric [building] :
```

```
==> Enter the label for this metric [building] :
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for it <y/n>? [n] :
==> Do you want to add another metric <y/n>? [y] :

Return value(s) for the selected method :
[1]   //ns0:Record/floor <string>
[2]   //ns0:Record/office <string>

==> Enter the index [1-2] of metric to display : 1
==> Enter the name for this metric [floor] :
==> Enter the label for this metric [floor] :
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for it <y/n>? [n] :
==> Do you want to add another metric <y/n>? [y] :

Return value(s) for the selected method :
[1]   //ns0:Record/office <string>

==> Enter the index [1-1] of metric to display : 1
==> Enter the name for this metric [office] :
==> Enter the label for this metric [office] :
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for it <y/n>? [n] :

Setup request parameters


==> Do you want to add User/Password Credential <y/n>? [n] : y


==> Do you want to add SSL TrustStore Credential <y/n>? [n] :


==> Do you want to add SSL KeyStore Credential <y/n>? [n] :


==> Is this metric group for periodic collection <y/n>? [y] :
The following units are for collection frequency:
[1]   Min
[2]   Hr
[3]   Day

==> Enter the index [1-3] of unit for this collection : 1
==> Enter the frequency of collection in Min : 5


==> Do you want to add another metric group <y/n>? [n] : y

All resource paths available for monitoring :
[1]   /add
[2]   /lookup/list
[3]   /lookup/phone
[4]   /lookup/building/people
[5]   /db/count

==> Enter the index [1-5] to select : 3
* Selected Resource Path : /lookup/phone

All methods available from the selected path :
[1]   application/json[PhoneInfo] GET(name)
[2]   application/xml[PhoneInfo] GET(name)

==> Enter the index [1-2] to select : 1
* Selected Resource Method: application/json[PhoneInfo] GET(name)
```

```
Define new metric group :
==> Enter the name for this metric group [GET] : LookupGet_JSON

Return value(s) for the selected method :
[1]    $..name <string>
[2]    $..phone <string>

==> Enter the index [1-2] of metric to display : 1
==> Enter the name for this metric [name] :
==> Enter the label for this metric [name] :
==> Is this a key metric <y/n>? [n] : y
==> Do you want to add another metric <y/n>? [y] :

Return value(s) for the selected method :
[1]    $..phone <string>

==> Enter the index [1-1] of metric to display : 1
==> Enter the name for this metric [phone] :
==> Enter the label for this metric [phone] :
==> Is this a key metric <y/n>? [n] :
==> Do you want to create threshold for it <y/n>? [n] :

Setup request parameters
==> Enter value for query parameter "name" [%LookupGet_JSON.name0000%] :
Harry Smith


==> Do you want to add User/Password Credential <y/n>? [n] : y

==> Do you want to add SSL TrustStore Credential <y/n>? [n] :

==> Do you want to add SSL KeyStore Credential <y/n>? [n] :

==> Is this metric group for periodic collection <y/n>? [y] :
The following units are for collection frequency:
[1]    Min
[2]    Hr
[3]    Day

==> Enter the index [1-3] of unit for this collection : 1
==> Enter the frequency of collection in Min : 5

==> Do you want to add another metric group <y/n>? [n] :

Files Generated:
- Target Metadata file: /scratch/work/metadata/OfficeDirectory.xml
- Target Collection file: /scratch/work/metadata/
OfficeDirectoryCollection.xml
```

**Example: REST CLI-Generated Target Metadata**

```
<TargetMetadata META_VER="1.0" TYPE="OfficeDirectory">
  <Display>
    <Label NLSID="NLSID_OFFICE_DIRECTORY">OfficeDirectory</Label>
    <ShortName NLSID="NLSID_OFFICE_DIRECTORY">OfficeDirectory</ShortName>
    <Description NLSID="NLSID_OFFICE_DIRECTORY">OfficeDirectory</Description>
  </Display>
  <Metric NAME="ListGet_XML" TYPE="TABLE">
    <Display>
      <Label NLSID="NLSID_LIST_GET_XML">ListGet_XML</Label>
```

```
          </Display>
      <TableDescriptor>
        <ColumnDescriptor IS_KEY="TRUE" NAME="name" TYPE="STRING">
          <Display>
            <Label NLSID="COL_NAME">name</Label>
          </Display>
        </ColumnDescriptor>
        <ColumnDescriptor IS_KEY="FALSE" NAME="title" TYPE="STRING">
          <Display>
            <Label NLSID="COL_TITLE">title</Label>
          </Display>
        </ColumnDescriptor>
        <ColumnDescriptor IS_KEY="FALSE" NAME="phone" TYPE="STRING">
          <Display>
            <Label NLSID="COL_PHONE">phone</Label>
          </Display>
        </ColumnDescriptor>
        <ColumnDescriptor IS_KEY="FALSE" NAME="building" TYPE="STRING">
          <Display>
            <Label NLSID="COL_BUILDING">building</Label>
          </Display>
        </ColumnDescriptor>
        <ColumnDescriptor IS_KEY="FALSE" NAME="floor" TYPE="STRING">
          <Display>
            <Label NLSID="COL_FLOOR">floor</Label>
          </Display>
        </ColumnDescriptor>
        <ColumnDescriptor IS_KEY="FALSE" NAME="office" TYPE="STRING">
          <Display>
            <Label NLSID="COL_OFFICE">office</Label>
          </Display>
        </ColumnDescriptor>
      </TableDescriptor>
      <QueryDescriptor FETCHLET_ID="JAXRS_Fetchlet">
        <Property NAME="ProxyHost" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyHost</
Property>
        <Property NAME="ProxyPort" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyPort</
Property>
        <Property NAME="SecurityPolicy" SCOPE="INSTANCE"
OPTIONAL="FALSE">ListGet_XML.SecurityPolicy</Property>
        <Property NAME="BaseURI" SCOPE="INSTANCE"
OPTIONAL="FALSE">ListGet_XML.BaseURI</Property>
        <Property NAME="Namespace" SCOPE="GLOBAL" OPTIONAL="FALSE"><!
[CDATA[[ns0="urn:com.office.directory"]]]></Property>
        <Property NAME="RowType" SCOPE="GLOBAL"
OPTIONAL="FALSE">//ns0:Record/name,//ns0:Record/title,//ns0:Record/phone,//
ns0:Record/building,//ns0:Record/floor,//ns0:Record/office</Property>
        <Property NAME="ColType" SCOPE="GLOBAL"
OPTIONAL="FALSE">name:STRING,title:STRING,phone:STRING,building:STRING,
floor:STRING,office:STRING</Property>
        <Property NAME="RequestMetadata" SCOPE="GLOBAL" OPTIONAL="FALSE"><!
[CDATA[<Resource path="/">
    <Resource path="lookup/list">
        <Method elementDefined="false" accept="application/xml" name="GET"/>
    </Resource>
</Resource>
]]></Property>
        <Property NAME="UserCredential" SCOPE="GLOBAL"
OPTIONAL="FALSE">UserCredentialSet</Property>
        <CredentialRef NAME="UserCredentialSet">UserCredentialSet</
CredentialRef>
```

```
          </QueryDescriptor>
        </Metric>
        <Metric NAME="LookupGet_JSON" TYPE="TABLE">
          <Display>
            <Label NLSID="NLSID_LOOKUP_GET_JSON">LookupGet_JSON</Label>
          </Display>
          <TableDescriptor>
            <ColumnDescriptor IS_KEY="TRUE" NAME="name" TYPE="STRING">
              <Display>
                <Label NLSID="COL_NAME">name</Label>
              </Display>
            </ColumnDescriptor>
            <ColumnDescriptor IS_KEY="FALSE" NAME="phone" TYPE="STRING">
              <Display>
                <Label NLSID="COL_PHONE">phone</Label>
              </Display>
            </ColumnDescriptor>
          </TableDescriptor>
          <QueryDescriptor FETCHLET_ID="JAXRS_Fetchlet">
            <Property NAME="ProxyHost" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyHost</
Property>
            <Property NAME="ProxyPort" SCOPE="INSTANCE" OPTIONAL="TRUE">ProxyPort</
Property>
            <Property NAME="SecurityPolicy" SCOPE="INSTANCE"
OPTIONAL="FALSE">LookupGet_JSON.SecurityPolicy</Property>
            <Property NAME="BaseURI" SCOPE="INSTANCE"
OPTIONAL="FALSE">LookupGet_JSON.BaseURI</Property>
            <Property NAME="Namespace" SCOPE="GLOBAL" OPTIONAL="FALSE"><!
[CDATA[[ns0="urn:com.office.directory"]]]></Property>
            <Property NAME="RowType" SCOPE="GLOBAL"
OPTIONAL="FALSE">$..name,$..phone</Property>
            <Property NAME="ColType" SCOPE="GLOBAL"
OPTIONAL="FALSE">name:STRING,phone:STRING</Property>
            <Property NAME="RequestMetadata" SCOPE="GLOBAL" OPTIONAL="FALSE"><!
[CDATA[
<Resource path="/">
    <Resource path="lookup/phone">
        <Method elementDefined="false" accept="application/xml" name="GET">
            <Parameter style="query" value="Harry Son" name="name"/>
        </Method>
    </Resource>
</Resource>
]]></Property>
            <Property NAME="UserCredential" SCOPE="GLOBAL"
OPTIONAL="FALSE">UserCredentialSet</Property>
            <CredentialRef NAME="UserCredentialSet">UserCredentialSet</
CredentialRef>
          </QueryDescriptor>
        </Metric>
        <CredentialInfo>
          <CredentialType NAME="CSFKeyCredential">
            <Display>
              <Label NLSID="CRED_TYPE">CSF-Key Credential Type</Label>
            </Display>
            <CredentialTypeColumn NAME="CSFKey">
              <Display>
                <Label NLSID="CRED_CSFKEY">Alias CSF Key</Label>
              </Display>
            </CredentialTypeColumn>
          </CredentialType>
          <CredentialType NAME="AliasCredential">
```

```xml
        <Display>
          <Label NLSID="CRED_TYPE">Alias Credential Type</Label>
        </Display>
        <CredentialTypeColumn NAME="Alias">
          <Display>
            <Label NLSID="CRED_ALIAS">Alias (i.e. username, encryption key,
signature key, etc)</Label>
          </Display>
        </CredentialTypeColumn>
        <CredentialTypeColumn NAME="Password">
          <Display>
            <Label NLSID="CRED_PASSWORD">Password for the alias</Label>
          </Display>
        </CredentialTypeColumn>
      </CredentialType>
      <CredentialSet NAME="UserCredentialSet" USAGE="MONITORING">
        <AllowedCredType TYPE="CSFKeyCredential"/>
        <AllowedCredType TYPE="AliasCredential"/>
      </CredentialSet>
    </CredentialInfo>
    <InstanceProperties>
      <InstanceProperty NAME="ProxyHost" CREDENTIAL="FALSE" OPTIONAL="TRUE">
        <Display>
          <Label NLSID="PROP_PROXY_HOST">Proxy Server Name</Label>
        </Display>
      </InstanceProperty>
      <InstanceProperty NAME="ProxyPort" CREDENTIAL="FALSE" OPTIONAL="TRUE">
        <Display>
          <Label NLSID="PROP_PROXY_PORT">Proxy Server Port</Label>
        </Display>
      </InstanceProperty>
      <InstanceProperty NAME="ListGet_XML.SecurityPolicy"
        CREDENTIAL="FALSE" OPTIONAL="FALSE">
        <Display>
          <Label NLSID="PROP_LIST_GET_XML_SECURITY_POLICY">[ListGet_XML]
Authentication/Web Service Policy</Label>
        </Display>
      </InstanceProperty>
      <InstanceProperty NAME="ListGet_XML.BaseURI" CREDENTIAL="FALSE"
OPTIONAL="FALSE">
        <Display>
          <Label NLSID="PROP_LIST_GET_XML_BASE_URI">[ListGet_XML] Resource
Base URI</Label>
        </Display>
      </InstanceProperty>
      <InstanceProperty NAME="LookupGet_JSON.SecurityPolicy"
        CREDENTIAL="FALSE" OPTIONAL="FALSE">
        <Display>
          <Label NLSID="PROP_LOOKUP_GET_JSON_SECURITY_POLICY">[LookupGet_JSON]
Authentication/Web Service Policy</Label>
        </Display>
      </InstanceProperty>
      <InstanceProperty NAME="LookupGet_JSON.BaseURI" CREDENTIAL="FALSE"
OPTIONAL="FALSE">
        <Display>
          <Label NLSID="PROP_LOOKUP_GET_JSON_BASE_URI">[LookupGet_JSON]
Resource Base URI</Label>
        </Display>
      </InstanceProperty>
    </InstanceProperties>
</TargetMetadata>
```

# 21

# Enterprise Manager DTD

A DTD provides the grammar for the XML files, thus describing what content is expected in each of its related XML files. When creating a new XML file, you need to carefully study its DTD to understand what content needs to be present in that file.

This chapter to provides a lookup of DTD elements to facilitate integration with Oracle Server Technology products:

- Terminology
- Target Metadata DTD Elements
- Target Collection DTD Elements

## Terminology

This chapter provides a lookup of DTD elements to facilitate integration with Oracle Server Technology products. Most of the examples in the document are snippets of an XML file.

**Target**: Target is a managed entity. A managed entity can be a hardware device or a software resource. Examples of a target are: host system, Oracle database, SMTP service etc.

**Associated Target**: Targets whose data depend on each other.

**Metric**: Collectable data

**Mid Tier: OMS** – Oracle Management Server

**Container**: A container is an entity that houses an Oracle installation. Currently two kinds of containers are possible – Oracle Home (Database, Enterprise Manager, Oracle Application Server installs) and ApplTop (application installs).

**Cluster Targets**: Cluster targets span across many hosts. All cluster targets represent the same target. The agents monitoring each of the cluster targets will produce the same metric result, severities, and so on.

**Cluster Interfaces**: Standard interfaces that the agent uses to talk to the cluster target.

## Target Metadata DTD Elements

This section defines DTD elements used by Enterprise Manager.

## TargetMetadata

The TargetMetadata describes the metadata for a target type. Metadata for a target describes its measurable characteristics, format of the collected data and the mechanism to collect or compute that data.

```
<!ELEMENT TargetMetadata (Display*, TypeProperties?, AssocTarget*,
DiscoveryHelper?, MonitoringMode*, AltSkipCondition?, MetricClass*,
Metric*, CredentialInfo?, InstanceProperties?, SSH_ERROR_MSG?)>

<!ATTLIST TargetMetadata

META_VER CDATA #REQUIRED

TYPE CDATA #REQUIRED

REQUIRED_AGENT_VERSION CDATA #IMPLIED

HELPID CDATA #IMPLIED

HELP CDATA #IMPLIED

CATEGORY_PROPERTIES CDATA #IMPLIED

RESOURCE_BUNDLE_PACKAGE CDATA #IMPLIED

TARGET_TYPE_CATEGORY CDATA #IMPLIED

HOST_BINDING_REQUIRED (TRUE | FALSE) "TRUE"

>
```

> **✎ Note:**
>
> The maximum length allowed for the various attributes mentioned in the
> file must be mentioend in tagsize.properties. This will be used to truncate
> the length of the attributes in the metadata file while loading the metadata
> information into the repository.

## Attributes

META_VER: Describes the version of metadata.

TYPE: Specifies the Target type.

HELPID: Not used.

HELP: Not used.

CATEGORY_PROPERTIES: Semicolon separated list of properties, used as
properties for ValidIf. Currently, each target type can have up to 5 properties used
as category property. EMAgent evaluates the values of the category properties and
makes it available within the metadata.

RESOURCE_BUNDLE_PACKAGE:

REQUIRED_AGENT_VERSION: This attribute indicates the minimum agent version
for the metadata. TargetMetadata marked with this attribute will be valid on Agent
versions greater than or equal to the specified version.

TARGET_TYPE_CATEGORY: Determines the Category to which the target type
belongs to. Multiple Target Type Categories can be specified as semicolon seperated
list of Categories.

HOST_BINDING_REQUIRED: Indicates whether the target needs a host target.

(TRUE (default) | FALSE)

## Elements

Display

TypeProperties

AssocTarget

DiscoveryHelper

MonitoringMode

AltSkipCondition

MetricCategory

Metric

CredentialInfo

InstanceProperties

SSH_ERROR_MSG

## Used In

TargetMetadata is a top-level element.

## Examples

```
<TargetMetadata TYPE="example1" META_VER="2.0"
REQUIRED_AGENT_VERSION="10.2.0.1.0">
```

. . .

```
</TargetMetadata>
```

The Metadata in the above example has REQUIRED_AGENT_VERSION attribute set to "10.2.0.1.0". This metadata will be valid only on agent versions 10.2.0.1.0 and higher.

```
<TargetMetadata TYPE="example1" META_VER="2.0">
```

```
<Metric NAME="prop" TYPE="TABLE">
```

```
<TableDescriptor>
```

```
<ColumnDescriptor NAME="name" TYPE="STRING" IS_KEY="TRUE" />
```

```
<ColumnDescriptor NAME="value" TYPE="STRING" />
```

```
</TableDescriptor>
```

```
<QueryDescriptor FETCHLET_ID="OS">
```

```
<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>
```

```
</QueryDescriptor>
```

```
</Metric>
```

```
</TargetMetadata>
```

This is a very simple example that describes a Target type, 'exmaple1' having data (Metric) that needs to be collected in the following format. The quoted values are evaluated by the 'OS' Fetchlet in accordance with the scoping rules defined in Property.

**Table 21-1    Metric Prop**

| Name | Value |
|------|-------|
| Host Name | NAME |

```
<TargetMetadata TYPE="example2" META_VER="2.0">
```

```
<Metric NAME="perf" TYPE="TABLE">
```

```
<TableDescriptor>
```

```
<ColumnDescriptor NAME="char" TYPE="STRING" IS_KEY="TRUE" />
```

```
<ColumnDescriptor NAME="value" TYPE="STRING" />
```

```
</TableDescriptor>
```

```
<QueryDescriptor FETCHLET_ID="OSLineToken" >
```

```
<Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl %scriptsDir%/
example1/perf.pl %port% </Property>
```

```
<Property NAME="delimiter" SCOPE="GLOBAL">=</Property>
```

```
<Property NAME="port" SCOPE="INSTANCE">accessPort</Property>
```

```
</QueryDescriptor>
```

```
</Metric>
```

```
<InstanceProperties>
```

```
<InstanceProperty NAME="accessPort" />
```

```
</InstanceProperties>
```

```
</TargetMetadata>
```

This sample illustrates the use of InstanceProperties. `InstanceProperties` element associates 'accessPort' to be associated with the target instance. The metric 'perf' is collected as shown below. The quoted values are evaluated by the 'OSLineToken' Fetchlet in accordance with the scoping rules defined in Property.

**Table 21-2    Metric: perf**

| Char | Value |
|------|-------|
| Command | '%perlBin%/perl %scriptsDir%/example1/perf.pl %port%' |

**Table 21-2 (Cont.) Metric: perf**

| Char | Value |
|------|-------|
| Delimiter | = |
| Port | 'accessPort' |

```
<TargetMetadata TYPE="example3" META_VER="2.0" CATEGORY_PROPERTIES="OS">

<Metric NAME="prop" TYPE="TABLE">

. . .

</Metric>

<InstanceProperties>

<DynamicProperties NAME="VersionAndLocation" FORMAT="ROW"
PROP_LIST="OS;OracleHome;Version">

<QueryDescriptor FETCHLET_ID="OSLineToken">

<Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property>

<Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>

<Property NAME="ENVEmdOS" SCOPE="SYSTEMGLOBAL">_emdOS</Property>

<Property NAME="ENVVersion" SCOPE="SYSTEMGLOBAL">_emdVersion</Property>

<Property NAME="ENVORACLE_HOME" SCOPE="SYSTEMGLOBAL">emdRoot</Property>

<Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl</Property>

<Property NAME="script" SCOPE="GLOBAL">%scriptsDir%/emdlocandver.pl </
Property>

<Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>

<Property NAME="delimiter" SCOPE="GLOBAL">|</Property>

</QueryDescriptor>

</DynamicProperties>

</InstanceProperties>

</TargetMetadata>
```

This sample uses a `DynamicProperties` element to return OS, OracleHome and Version properties. The scripts return results that are parsed to return the properties listed in the PROP_LIST attribute.

## Display

Specifies the information to be used by the Console UI for displaying the element that has this tag.

```
<!ELEMENT Display (ValidIf*, Label, ShortName?, Icon?, Description?,
Unit?)>

<!ATTLIST Display

FOR_SUMMARY_UI (TRUE | FALSE) "FALSE">
```

Metric owners needs to follow the same metric metadata registration mechanism to add units and unit category to their metrics. The standard metric units are as follows:

- Adding units and unit category

```
<ColumnDescriptor NAME="Network_Sent_Rate" TYPE="NUMBER">
<Display>
<Unit>KBPS</Unit>
<UnitCategory>RATE<UnitCategory>
</Display>
</ColumnDescriptor>
```

- Adding Metrics with no units

```
<ColumnDescriptor NAME="Number of Requests" TYPE="NUMBER">
<Display>
<Unit>NA</Unit> -- Counter Metric, No Unit
<UnitCategory>COUNT<UnitCategory>
</Display>
</ColumnDescriptor>

<ColumnDescriptor NAME="Service specified by the local port" TYPE="NUMBER">
<Display>
<Unit>NA</Unit> -- PORT, No Unit
<UnitCategory>PORT</UnitCategory>
</Display>
</ColumnDescriptor>
```

For more information on standard metric units, see Metric Unit Standardization.

## Attributes

FOR_SUMMARY_UI: Indicates whether a column is visible in the condensed UI. Condensed view is necessary when all the columns cannot fit in one UI page. In a condensed view, only columns whose FOR_SUMMARY_UI=TRUE will be displayed.

TRUE | FALSE (default)

## Elements

ValidIf

Label

ShortName

Icon

Description

Unit

## Used In

TargetMetadata

Metric

ColumnDescriptor

CredentialType

CredentialTypeColumn

CredentialSet

CredentialSetColumn

InstanceProperty

## Examples

Display element must contain a Label element. The elements ShortName, Icon, Description and Unit are optional. If ValidIf element(s) are present then all the conditions in the ValidIf elements must be satisfied for the element to be displayed.

```
<Display FOR_SUMMARY_UI="TRUE">

<Label NLSID="emd_resp_stat">Status</Label>

</Display>
```

This example describes the display characteristics for the element that includes it. The 'FOR_SUMMARY_UI' set to TRUE forces the UI to display the element in the condensed view also.

```
<Display FOR_SUMMARY_UI="TRUE">

<ValidIf>

<CategoryProp NAME="OS" CHOICES="SunOS"/>

</ValidIf>

<Label NLSID="id_name_for_agent">Agent Name</Label>

<ShortName NLSID="id_short_name">Name</ShortName>

<Icon GIF="a.gif">Name</Icon>

<Description NLSID="id_for_description">Displays the Agent Name</Description>

</Display>
```

This sample describes the display characteristics for an element that are valid only for 'SunOS' OS.

# SSH_ERROR_MSG

Free form error message to be displayed at the console in case SSH is not available for monitoring a remote target of this type.

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

<!ATTLIST SSH_ERROR_MSG

NLSID CDATA #REQUIRED

>
```

## Attributes

NLSID: The NLSID of the free form error message

## Elements

None.

## Used In

TargetMetadata

## Examples

```
<SSH_ERROR_MSG NLSID="ssh_error_nlsid">

Target data will not be available until sshd is available

</SSH_ERROR_MSG>
```

# TypeProperties

The TypeProperties holds TypeProperty elements for the target type.

```
<!ELEMENT TypeProperties (TypeProperty*)>
```

## Attributes

None.

## Elements

TypeProperty

## Used In

TargetMetadata

## Examples

```
<TypeProperties>

<TypeProperty PROPERTY_NAME="a_name" PROPERTY_VALUE="a_value"/>

</TypeProperties>
```

The property name-value pair, a_name,a_value, would apply to the target type.

# TypeProperty

TypeProperty element contains the property name-value pair for a target type.

```
<!ELEMENT TypeProperty EMPTY>

<!ATTLIST TypeProperty

PROPERTY_NAME CDATA #REQUIRED

PROPERTY_VALUE CDATA #IMPLIED

>
```

## Attributes

PROPERTY_NAME: Name of the target type property.

PROPERTY_VALUE: Value of the target type property.

## Elements

None.

## Used In

TypeProperties

## Examples

Refer to the example for TypeProperties.

# AssocTarget

The AssocTarget describes how two targets are related to each other. Targets may be associated for a number of reasons some of them being: rendering topology maps, root cause analysis, determining availability of targets, minimizing redundancy in data collection and transmission and determining order for data collection or job execution amongst others. For instance, if a fault occurs on a target, its associated target might be affected too. This information would be valuable in root cause analysis.

```
<!ELEMENT AssocTarget (AssocPropDef*)>

<!ATTLIST AssocTarget

ASSOC_TARGET CDATA #IMPLIED

TYPE CDATA #IMPLIED

ASSOCIATION_NAME CDATA #IMPLIED

NAME_NLSID CDATA #IMPLIED

DESCRIPTION CDATA #IMPLIED

DESCRIPTION_NLSID CDATA #IMPLIED
```

```
SOURCE_TARGET_TYPE CDATA #IMPLIED

ASSOC_TARGET_TYPE CDATA #IMPLIED

CARDINALITY (OPTIONAL_SINGLE_CARDINAL |

REQUIRED_SINGLE_CARDINAL |

OPTIONAL_MULTI_CARDINAL |

REQUIRED_MULTI_CARDINAL) #IMPLIED

ASSOC_TYPE (RELATES_TO|DEPENDS_ON | CONNECTS_TO |

SERVICE_ACCESS_POINT|RUNS_ON|

CONTAINS|HOSTED_BY|MONITORED_BY|

OPTIONALLY_CONNECTS_TO) #IMPLIED

COMPUTE_RULE (PARENT | MEMBER | NONE) "NONE"

>
```

## Attributes

ASSOC_TARGET: the name of the associated target.

TYPE: the target type of the associated target.

ASSOCIATION_NAME: deprecates ASSOC_TARGET. Specifies the name of the association

ASSOC_TARGET_TYPE: deprecates TYPE. Specifies the target type that is associated with this target. 'ANY' can be used to indicate that the association target could be any target type.

NAME_NLSID: NLSID for association name

DESCRIPTION: Description

DESCRIPTION_NLSID: NLSID for description string

SOURCE_TARGET_TYPE: If association starts from a target, other than the target itself, the target type of the source target is specified in this attribute. 'ANY' can be used to indicate that the source could be any target type.

CARDINALITY: Specifies the cardinality of the associated targets. Supported values are:

1. OPTIONAL_SINGLE_CARDINAL: zero or one targets as associated
2. REQUIRED_SINGLE_CARDINAL: exactly one associated target
3. OPTIONAL_MULTI_CARDINAL: zero or several associated targets
4. REQUIRED_MULTI_CARDINAL: one or more associated targets

ASSOC_TYPE: Describes the relation of the associated targets. Supported Values are:

1. RELATES_TO (default): implies "some" generic relationship
2. DEPENDS_ON: Dependency on associated target

3. CONNECTS_TO: Source target connects to assoc target

4. SERVICE_ACCESS_POINT

5. RUNS_ON: Source target runs on (installed on) assoc target

6. CONTAINS: Source target contains assoc target

7. HOSTED_BY: Similar to runs on

8. MONITORED_BY: Source target is monitored by assoc target (agent)

9. OPTIONALLY_CONNECTS_TO

COMPUTE_RULE: Describes how the Association is computedIif not already present) for the target instance. Supported values are:

1. PARENT: will be constructed based on the first parent of the target.

2. MEMBER: will be construced based on the first child of the target.

3. NONE (default): Association for the target instance is not automatically computed.

## Elements

AssocPropDef (Not supported in version 10.2.)

## Used In

TargetMetadata

## Examples

AssocTarget element for versions prior to 10.2 MUST have the following attributes:

1. ASSOC_TARGET

2. TYPE

AssocTarget element for versions 10.2 and later MUST use the following attributes at least.

1. ASSOCIATION_NAME instead of ASSOC_TARGET.

2. ASSOC_TARGET_TYPE instead of TYPE.

```
<TargetMetadata TYPE="oracle_email" META_VER="2.0">

<AssocTarget ASSOCIATION_NAME="IM"

SOURCE_TARGET_TYPE="oracle_email"

ASSOC_TARGET_TYPE="oracle_im"

ASSOCIATION_TYPE="DEPENDS_ON"

NAME_NLSID="im_assoc_name"

DESCRIPTION="This association captures Email-IM dependency"

DESCRIPTION_NLSID="im_assoc_description" />

. . .
```

```
</TargetMetadata>
```

This element would be defined in 'oracle_email' and would represent the following relation:

Oracle_email -----DependsOn----à oracle_im

# AssocPropDef

The AssocPropDef describe the properties for an association. This element is not supported in version 10.2.

```
<!ELEMENT AssocPropDef EMPTY>

<!ATTLIST AssocPropDef

NAME CDATA #REQUIRED

REQUIRED (TRUE | FALSE) #REQUIRED>
```

## Attributes

NAME: Name of the property.

REQUIRED: Indicates whether the property is required.

TRUE | FALSE

## Elements

None.

## Used In

AssocTarget

## Examples

This element is not supported.

# DiscoveryHelper

The DiscoveryHelper helps the agent in its process of discovering the target type.

```
<!ELEMENT DiscoveryHelper (DiscoveryHint*) >

<!ATTLIST DiscoveryHelper

CATEGORYNAME CDATA #REQUIRED

OUI_BASED (TRUE | FALSE) "TRUE"

>
```

## Attributes

CATEGORYNAME: name of the category in discover.lst which discovers targets for a given type.

OUI_BASED: boolean value to indicate if this discovery used OUI inventory info.

TRUE (default) | FALSE

## Elements

DiscoveryHint

## Used In

TargetMetadata

# DiscoveryHint

The DiscoveryHint allows users to specify any hint which can be a guide to discovery.

```
<!ELEMENT DiscoveryHint (Display?) >

<!ATTLIST DiscoveryHint

NAME CDATA #REQUIRED

>
```

## Attributes

NAME: name of the hint to guide discovery process

## Elements

Display

## Used In

DiscoveryHelper

# MetricClass

MetricClass provides a means for classifying Metrics into categories. Metrics can be classified into categories based on multiple characteristics such as Function (Perf, Load, Config), EvaluationCost (Cheap, Medium, Expensive) and Applicability (Typical, Esoteric).

```
<!ELEMENT MetricClass (MetricCategory*)>

<!ATTLIST MetricClass

NAME CDATA #REQUIRED

NLSID CDATA #IMPLIED>
```

## Attributes

NAME: Is the name of the class (e.g. Functional)

NLSID: Is the translation ID. The naming convention is metric_class_<classname>

## Elements

MetricCategory

## Used In

TargetMetadata

## Examples

```
<TargetMetadata TYPE="example3" META_VER="2.0">

<MetricClass NAME="EvaluationCost" NLSID="id_for_eval_cost_class">

<MetricCategory NAME="CHEAP" NLSID="id_for_cheap_cat"/>

<MetricCategory NAME="MEDIUM" NLSID="id_for_medium_cat"/>

<MetricCategory NAME="EXPENSIVE" NLSID="id_for_expensive_cat"/>

</MetricClass>

<Metric NAME="metric1" TYPE="TABLE">

<CategoryValue Class="EvaluationCost" CATEGORY_NAME="CHEAP"/>

. . .

</Metric>

. . .

</TargetMetadata>
```

The example describes adding a 'EvaluationCost' MetricClass for a Target type 'example3'. EvaluationCost has 3 categories: CHEAP, MEDIUM, and EXPENSIVE. Metric, 'metric1' is a CHEAP metric to evaluate.

Refer to the explanations of CategoryValue, Metric for more details.

# MetricCategory

A MetricCategory element lists each choice within a classification of metrics.

```
<!ELEMENT MetricCategory EMPTY>

<!ATTLIST MetricCategory

NAME CDATA #REQUIRED

NLSID CDATA #IMPLIED>
```

## Attributes

NAME: The name of the category (e.g. Security)

NLSID: The NLSID of the category. The naming convention here is metric_cat_<category_name>

## Elements

None.

## Used In

MetricClass

## Examples

Refer to the example for MetricClass.

# Metric

A metric element is used to declare the different measurable characteristics (performance, load, configuration and so on) of a target. The metric element describes the structure of the collected data as well as how to compute the information.

> **Note:**
>
> Oracle recommends that every target type have a special metric named "Response". This metric should have a column called "Status". The type creator should also set up a collection of this metric and set up a condition (see TargetCollection.dtd) on the Status column. The availability system (target up/down status over time) uses alerts on this metric column to provide up/down statistics over time.

**Note**: QueryDescriptor is required for Management Repository metrics.

```
<!ELEMENT Metric ((ValidIf | ValidMidTierVersions)*, Display?,
CategoryValue* ,TableDescriptor?, ((QueryDescriptor | ExecutionDescriptor)
| PushDescriptor)* )>

<!ATTLIST Metric

NAME CDATA #REQUIRED

TYPE (NUMBER | STRING | TABLE | RAW | EXTERNAL | REPOSITORY_TABLE |
REPOSITORY_NUMBER | REPOSITORY_STRING | REPOSITORY_EVENT) "NUMBER"

REPOSITORY (TRUE|FALSE) "FALSE"

USAGE_TYPE (VIEW_COLLECT | REALTIME_ONLY | HIDDEN | HIDDEN_COLLECT |
COLLECT_UPLOAD) "VIEW_COLLECT"
```

```
KEYS_FROM_MULT_COLLS (TRUE | FALSE) "FALSE"

IS_TEST_METRIC (TRUE | FALSE) "FALSE"

KEYS_ONLY (TRUE | FALSE) "FALSE"

REMOTE (TRUE | FALSE) "FALSE"

IS_TRANSPOSED (TRUE | FALSE) "FALSE"

HELP CDATA #IMPLIED

IS_METRIC_LONG_RUNNING (TRUE|FALSE) "FALSE"

CONFIG (TRUE|FALSE) "FALSE"

FORCE_CACHE (TRUE | FALSE) "FALSE"

COLLECT_ON_ALL_NODES (TRUE | FALSE) "FALSE"

INCREMENTAL (TRUE|FALSE) "FALSE"

NUM_CACHE_VALUES CDATA "1"

LOCAL_ONLY (TRUE | FALSE) "FALSE"

>
```

## Attributes

NAME: Specifies Metric name, it uniquely identifies it within the scope of its target type.

TYPE: Specifies the data type. Supported metric types are:

a) NUMBER (default): Deprecated – Instead use a table with 1 column of type NUMBER.

b) STRING: Deprecated – Instead use a table with 1 column of type STRING

c) TABLE: Tabular data

d) RAW: Tabular data

e) EXTERNAL: Data not parsed/fronted by EMD.

f) REPOSITORY_EVENT

REPOSITORY: This attribute indicates a metric that will be collected by the Management Repository. This is a boolean attribute.

FALSE (default) indicates that the metric is collected by the Management Agent.

TRUE indicates that the metric is collected at the Management Repository. When this attribute is set to TRUE, the query descriptor must be similar to the following:

```
<QueryDescriptor FETCHLET_ID="REPOSITORY_SQL">
              <Property NAME="Type">SQL</Property>
              <Property NAME="Source">CDATA</Property>
          </QueryDescriptor>
```

There must be one query descriptor only. Possible values for Type are:

- SQL

- PLSQL

- BULK_PLSQL

> **Note:**
>
> When REPOSITORY is set to TRUE, the Metric TYPE must be TABLE.
> RAW is not supported for Repository metrics.

USAGE_TYPE: This defines the purpose of the metric. Supported types are:

a) VIEW_COLLECT *(default):* These are metrics that are both viewable and collected.

b) REALTIME_ONLY: These are metrics that cannot be collected. The rules on key uniqueness are not applied to these metrics.

c) HIDDEN: Metrics are tagged hidden when they shouldn't be collected nor visible from the console. The data is not uploaded either. These are "temporary" metrics used to compute other metrics.

d)HIDDEN_COLLECT: Metric can be collected. It will not be viewable. The data is not uploaded. It is similar to HIDDEN, but collection criteria can be defined for it.

e) COLLECT_UPLOAD: Metric can be collected and uploaded, Metadata is uploaded to MGMT_METRICS but it is not viewable in All Metrics page.

Mapping of old USAGE_TYPE values

DISPLAY_ONLY: REALTIME_ONLY

MULTI_KEY: VIEW_COLLECT

COLLECT_ONLY: VIEW_COLLECT

The metric browser automatically decides which metrics to not display.

KEYS_FROM_MULT_COLLS: If TRUE, the attribute indicates that there are multiple key columns. The combination of key columns uniquely identifies a row. If the value is TRUE only then can the metric be collected in multiple collection items.

TRUE | FALSE (default)

IS_TEST_METRIC: The agent can check some metrics to determine if a target has been correctly specified with valid instance properties. This attribute marks this metric as one of the test metrics.

TRUE or FALSE (default)

HELP: Help text – This attribute is not used.

KEYS_ONLY: It is used to tag special metrics that have only key columns. Note that in general, such metrics are not useful in collections (since no data is uploaded), but there may be special cases where the metric is used to just retrieve a set of keys.

TRUE or FALSE (default)

IS_METRIC_LONG_RUNNING: IF true, the metric is long running. This gives the metric engine, a hint that this query will take relatively longer to finish. A special property EM_IS_METRIC_LONG_RUNNING will be passed to fetchlet automatically.

TRUE or FALSE (default)

CONFIG: This is a special designation for CONFIG metrics that are uploaded differently by the Enterprise Manager framework.

TRUE or FALSE (default)

REMOTE: It is used to tag metrics that can be evaluated from a remote location. These metrics could be evaluated from "beacon" nodes

IS_TRANSPOSED: It is used to tag metrics that generate data as name value pairs and the UI treats the names as "column headers". These are useful when the number of rows (or data categories) is not known at design time.

FORCE_CACHE: For collected metrics, this is a strong hint to the agent to cache the results of a metric collection. In the absence of this hint, the agent may only start caching the result of a metric after it realizes that someone will try to use the cached value.

COLLECT_ON_ALL_NODES: For a clustered target, metrics marked with this attribute set to TRUE, will be collected on all the nodes of the cluster.

TRUE | FALSE (default)

INCREMENTAL: This attribute is used ONLY by the OCM collector. This attribute is TRUE iff the metric is incremental i.e. the rows collected during a collection do not replace the ones collected during the previous collection but rather add to them. This is the case for metric data whose lifespan extend across collections for example the ECM_RUNNING_PRODUCTS metric.

NUM_CACHE_VALUES: Starting with 11, the agent will support the ability to cache multiple collection results in memory for access by the EMDClient getMetricHistory API. The value of this attribute defaults to 1, but the user can set it to a higher value such as 15 or 60

LOCAL_ONLY: Specifies that a metric should be collected for local targets only and skipped for remote targets.

## Elements

ValidIf

ValidMidTierVersions

Display

CategoryValue

TableDescriptor

QueryDescriptor

ExecutionDescriptor

PushDescription

## Used In

TargetMetadata

## Examples

```
<TargetMetadata TYPE="example1" META_VER="2.0"
CATEGORY_PROPERTIES="OS;Version">

<Metric NAME="prop" TYPE="TABLE">

<Display>

<Label NLSID="example1_metric">Example1 Metric</Label>

</Display>

<TableDescriptor>

<ColumnDescriptor NAME="name" TYPE="STRING" IS_KEY="TRUE" />

<ColumnDescriptor NAME="value" TYPE="STRING" />

</TableDescriptor>

<QueryDescriptor FETCHLET_ID="OS">

<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>

</QueryDescriptor>

</Metric>

</TargetMetadata>
```

This is the most common form of a metric definition. The statement declares the metric, 'example1', to contain tabular data.

If a metric has a "TABLE" type, the value will be returned as a set of rows each containing a set of values (columns). A list will be a special case of the Table. A Table Metric must have a TableDescriptor defined.

```
<Metric NAME="Inventory" TYPE="EXTERNAL" >

<ValidIf>

<CategoryProp NAME="OS" CHOICES="SunOS"/>

</ValidIf>

<Display>

<Label NLSID="host_Inventory">Inventory</Label>

</Display>

<QueryDescriptor FETCHLET_ID="OS">

<Property NAME="emdRoot" SCOPE="SYSTEMGLOBAL">emdRoot</Property>

<Property NAME="emHome" SCOPE="SYSTEMGLOBAL">agentStateDir</Property>
```

```
<Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property>

<Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>

<Property NAME="hostConfigClasspath"
SCOPE="SYSTEMGLOBAL">hostConfigClasspath</Property>

<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>

<Property NAME="type" SCOPE="INSTANCE">TYPE</Property>

<Property NAME="display_target_name" SCOPE="INSTANCE">DISPLAY_NAME</
Property>

<Property NAME="display_target_type" SCOPE="INSTANCE">TYPE_DISPLAY_NAME</
Property>

<Property NAME="command" SCOPE="GLOBAL">"%perlBin%/perl"
"%scriptsDir%/osm/ecmCollectInventory.pl" "%hostConfigClasspath%"
"%perlBin%" "%emdRoot%" "%emHome%" "%hostname%" "%loaderFile%" "%emHome%/
sysman/config/OUIinventories.add" "%type%" "%display_target_name%"
"%display_target_type%"</Property>

</QueryDescriptor>

</Metric>
```

This example declares 'Inventory' as EXTERNAL which implies that the metric will be evaluated in the correct format and will be placed in the upload directory. The EMAgent will not parse the metric result. The ValidIf element ensures that the metric will be evaluated only for 'SunOS' OS.

```
<Metric NAME="AddressMap" TYPE="TABLE" FORCE_CACHE="TRUE">

The contents would be similar to a TABLE metric

</Metric>
```

In this example, the agent is forced to cache the results for the AddressMap metric.

```
<Metric NAME="ICMPPing" TYPE="TABLE" IS_TEST_METRIC="TRUE"
USAGE_TYPE="HIDDEN">

The contents would be similar to a TABLE metric

</Metric>
```

'ICMPPing' is identified as a test metric. The agent will use its value to verify that the target identified by the instance properties is correct. Since the purpose of this metric is for INTERNAL use only, it is marked as 'HIDDEN'.

USAGE_TYPE Summary:

| USAGE_TYPE | KEY_UNIQUE_CHECK | COLLECTABLE | MGMT_METRICS_RAW | MGMT_METRICS | VIEWABLE |
|---|---|---|---|---|---|
| VIEW_COLLECT | Y | Y | Y | Y | Y |
| REALTIME_ONLY | N | N | N | Y | N |
| HIDDEN | Y | N | N | Y | N |
| HIDDEN_COLLECT | Y | Y | N | Y | N |

| USAGE_TYPE | KEY_UNIQU E_CHECK | COLLECTAB LE | MGMT_MET RICS_RAW | MGMT_ME TRICS | VIEWABLE |
|---|---|---|---|---|---|
| COLLECT_UPLOA D | Y | Y | Y | Y | Y |

```
<Metric NAME="http_raw" TYPE="TABLE" KEYS_FROM_MULT_COLLS="TRUE"
REMOTE="TRUE">
```

```
The contents would be similar to a TABLE metric
```

```
</Metric>
```

'http_raw' metric can be evaluated from a remote location and therefore is tagged as 'REMOTE'.

```
<Metric NAME="openPorts" TYPE="RAW" CONFIG="TRUE" KEYS_ONLY="TRUE"
HELP="NO_HELP">
```

```
<ValidIf>
```

```
<CategoryProp NAME="OS" CHOICES="SunOS"/>
```

```
</ValidIf>
```

```
<Display>
```

```
<Label NLSID="host_open_ports_ESM">Open Ports</Label>
```

```
</Display>
```

```
<TableDescriptor TABLE_NAME="esm_collection">
```

```
<ColumnDescriptor NAME="property" COLUMN_NAME="property" TYPE="STRING"
IS_KEY="TRUE" HELP="NO_HELP"/>
```

```
<ColumnDescriptor NAME="value" COLUMN_NAME="value" TYPE="STRING"
IS_KEY="TRUE" HELP="NO_HELP"/>
```

```
</TableDescriptor>
```

```
<QueryDescriptor FETCHLET_ID="OSLineToken">
```

```
<Property NAME="scriptsDir" SCOPE="SYSTEMGLOBAL">scriptsDir</Property>
```

```
<Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>
```

```
<Property NAME="command" SCOPE="GLOBAL">%perlBin%/perl</Property>
```

```
<Property NAME="script" SCOPE="GLOBAL">%scriptsDir%/openports.pl</
Property>
```

```
<Property NAME="startsWith" SCOPE="GLOBAL">em_result=</Property>
```

```
<Property NAME="delimiter" SCOPE="GLOBAL">=</Property>
```

```
</QueryDescriptor>
```

```
</Metric>
```

'openPorts' is defined as a CONFIG metric. Since the type is RAW, EMAgent expects the values in the correct format.

```
<Metric NAME="storage_reporting_data" TYPE="RAW" CONFIG="TRUE"
IS_METRIC_LONG_RUNNING="TRUE">

The contents would be similar to a RAW metric

</Metric>
```

This specifies that storage_reporting_data takes a long time to execute.

# ValidIf

The ValidIf element is used to create type definitions that apply to multiple flavors of a target. To do this, certain properties of the target (up to a max of 5) can be marked as category properties, and ValidIf elements can be placed in portions of the metadata to indicate that they are only applicable if the target's property values match the specified values.

The CategoryProp elements within a ValidIf should all match for the containing element to be evaluated. A containing element may include multiple ValidIfs to indicate its applicability for different sets of conditions.

```
<!ELEMENT ValidIf (CategoryProp+)>
```

## Attributes

None.

## Elements

CategoryProp

## Used In

Metric

QueryDescriptor

Display

MonitoringMode

InstanceProperty

DynamicProperties

ExecutionDescriptor

PushDescription

CollectionItem

## Examples

```
<TargetMetadata TYPE="example1" META_VER="2.0"
CATEGORY_PROPERTIES="OS;Version">
```

```
<Metric NAME="prop" TYPE="TABLE">

<ValidIf>

<CategoryProp NAME="OS" CHOICES="SunOS"/>

<CategoryProp NAME="Version" CHOICES="5.9"/>

</ValidIf>

<TableDescriptor>

<ColumnDescriptor NAME="name" TYPE="STRING" IS_KEY="TRUE" />

<ColumnDescriptor NAME="value" TYPE="STRING" />

</TableDescriptor>

<QueryDescriptor FETCHLET_ID="OS">

<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>

</QueryDescriptor>

</Metric>

</TargetMetadata>
```

This example indicates that the category properties, 'OS' and 'Version' must have values, 'SunOS' and '5.9' for the metric, 'prop' to be evaluated. EMAgent allows the definition of upto 5 category properties which can be used in ValidIfs elements.

# CategoryProp

The CategoryProp element is used to list the allowed values for a property for the ValidIf to match.

```
<!ELEMENT CategoryProp EMPTY>

<!ATTLIST CategoryProp

NAME CDATA #REQUIRED

CHOICES CDATA #REQUIRED>
```

## Attributes

NAME: Specifies the name of the category property.

CHOICES: Specifies the values the property can have. This may contain values separated by ";".

## Elements

None.

## Used In

ValidIf

## Examples

```
<ValidIf>

<CategoryProp NAME="OS" CHOICES="SunOS"/>

<CategoryProp NAME="Version" CHOICES="5.8;5.9"/>

</ValidIf>
```

If the example described in the ValidIf statement is modified such that the 'Version' CategoryProperty now has two choices 5.8 and 5.9, the metric would be evaluated for 'SunOS' versions 5.8 and 5.9.

```
<ValidIf>

<CategoryProp NAME="OS" CHOICES="SunOS"/>

</ValidIf>

<ValidIf>

<CategoryProp NAME="OS" CHOICES="AIX"/>

</ValidIf>
```

A metric that has ValidIfs as shown above, would be evaluated if OS is either SunOS or AIX.

Note: If ValidIfs are used to differentiate between multiple definitions of the same metric, there should be no cases where multiple definitions of the metric get validated.

```
<Metric NAME="a_metric" TYPE="TABLE">

<ValidIf>

<CategoryProp NAME="C1" CHOICES="A;B">

</ValidIf>

. . .

</Metric>

<Metric NAME="a_metric" TYPE="TABLE">

<ValidIf>

<CategoryProp NAME="C1" CHOICES="A;C">

</ValidIf>

. . .

</Metric>
```

The above example demonstrates how NOT to use ValidIfs.

# ValidMidTierVersions

The ValidMidTierVersions element is used in the mid tier based versioning support in the agent.

This element can be used either under a Metric or within a CustomTableMapper element. When present, it indicates to the agent that a Metric definition or a CustomTableMapper definition only applies for a certain set of mid tier versions.

```
<!ELEMENT ValidMidTierVersions EMPTY>

<!ATTLIST ValidMidTierVersions

PLUG_IN CDATA #IMPLIED

START_VER CDATA #IMPLIED

END_VER CDATA #IMPLIED>
```

## Attributes

PLUG_IN: Optional attribute that allows a particular mid tier plug in to be referenced. If not specified, this tag applies to the core OMS version.

START_VER: Starting version (inclusive, optional) the element is applied from.

END_VER: Ending version (exclusion, optional) that the element is applicable to.

## Elements

None.

## Used In

Metric

CustomTableMapper

## Examples

Every ValidMidTierVersions element needs to have at least one of START_VER or END_VER specified.

```
<Metric NAME="metric1" TYPE="RAW">

<TableDescriptor>

<CustomTableMapper REP_TABLE_NAME="table1">

<ValidMidTierVersions PLUG_IN="DB" START_VER="10.1" END_VER="10.3"/>

<ColumnMapper METRIC_COLUMN="col1" REP_TABLE_COLUMN="col1_rep"/>

</CustomTableMapper>

<ColumnDescriptor NAME="c1" COLUMN_NAME="col1" TYPE="STRING"/>

<ColumnDescriptor NAME="c2" COLUMN_NAME="col2" TYPE="STRING"/>
```

```
</TableDescriptor>

<QueryDescriptor>

. . .

</QueryDescriptor>

</Metric>
```

The CustomTableMapper element mapping 'metric1' to repository table, 'table1' is applicable only for DB Plugin versions between 10.1 (inclusive) and 10.3 (not inclusive).

```
<ValidMidTierVersions START_VER="10.1.0.1"/>
```

This element if present in a Metric or CustomTableMapper would indicate to the EMAgent that the Metric or the CustomTableMapper is applicable only to OMS versions 10.1.0.1 and higher.

```
<ValidMidTierVersions END_VER="10.2"/>
```

This element if present in a Metric or CustomTableMapper would indicate to the EMAgent that the Metric or the CustomTableMapper is applicable only to OMS versions less than (not including) 10.2.

# TableDescriptor

TableDescriptor describes the structure of the data for the metric of type TABLE.

```
<!ELEMENT TableDescriptor (ColumnDescriptor+, CustomTableMapper*)>

<!ATTLIST TableDescriptor

TABLE_NAME CDATA #IMPLIED

SKIP_TARGET_COLUMN (TRUE | FALSE) "FALSE"

SKIP_METRIC_COLUMN (TRUE | FALSE) "FALSE"

SKIP_COLLTIME_PK (TRUE | FALSE) "FALSE"

SKIP_COLLTIME_COLUMN (TRUE | FALSE) "FALSE">
```

> **✎ Note:**
>
> SKIP_COLLTIME_PK attribute is deprecated. This attribute specifies that the collection timestamp should not be a part of the primary key. This was used to indicate that the latest row should override any previous rows with the same primary key. Since this can now be done by simply altering the table definition in the repository, SKIP_COLLTIME_PK is not useful any more.

## Attributes

TABLE_NAME: This attribute specifies the repository database table into which the collected data will be loaded to. Note: Only RAW metrics can define this attribute.

If a TableDescriptor contains CustomTableMapper elements, it should not contain a TABLE_NAME attribute.

SKIP_TARGET_COLUMN: This attribute is applicable for a RAW metric only. If set to TRUE, Target GUID column will not be generated.

TRUE | FALSE (default)

SKIP_METRIC_COLUMN: This attribute is applicable for a RAW metric only. If set to TRUE, the Metric Name column will not be generated.

TRUE | FALSE (default)

SKIP_COLLTIME_PK: Deprecated in version 10.2.The SKIP_COLLTIME_PK option can be used if the collection timestamp needs to be generated, but not added as a primary key.

TRUE | FALSE (default)

## Elements

ColumnDescriptor

CustomTableMapper

## Used In

Metric

## Examples

A Table Metric must have a TableDescriptor defined. It describes columns of the table.

```
<Metric NAME="prop" TYPE="TABLE">

<Display>

<Label NLSID="example1_metric">Example1 Metric</Label>

</Display>

<TableDescriptor>

<ColumnDescriptor NAME="name" TYPE="STRING" IS_KEY="TRUE" />

<ColumnDescriptor NAME="value" TYPE="STRING" />

</TableDescriptor>

<QueryDescriptor FETCHLET_ID="OS">

<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>

</QueryDescriptor>

</Metric>
```

This is the most common usage for the TableDescriptor element that represents a metric of TYPE=TABLE. For mid-tier versioning support refer to the Examples for CustomTableMapper.

```
<TableDescriptor TABLE_NAME="esm_collection">
```

This declaration is valid only for a RAW metric. This element contains the description of esm_collection table within a RAW metric.

```
<TableDescriptor TABLE_NAME="mgmt_db_compatibility"

SKIP_COLLTIME_PK="TRUE" SKIP_COLLTIME_COLUMN="TRUE"

SKIP_METRIC_COLUMN="TRUE" SKIP_TARGET_COLUMN="TRUE">
```

The SKIP attributes may be applied only to RAW metrics. The EMAgent automatically generates TARGET_GUID, METRIC_NAME and COLLECTION_TIMESTAMP columns for a raw metric unless explicitly indicated by setting SKIP attributes to TRUE. The TableDescriptor, in this sample explicitly requests the omission of the default columns.

# ColumnDescriptor

The ColumnDescriptor elements describe each column in a table. The agent also supports one level of nesting of tables for RAW metrics. This allows metrics to return a table of data in place of a column which is uploaded in the context of the containing row. For example: In a metric that returns a list of expensive SQL statements in a database, a column that returns the multi-row explain plan for the SQL statement could be returned in a nested raw metric column.

```
<!ELEMENT ColumnDescriptor (Display?, CategoryValue*, TableDescriptor?)>

<!ATTLIST ColumnDescriptor

NAME CDATA #REQUIRED

TYPE (NUMBER | STRING | RAW | CLOB | BLOB) "NUMBER"

IS_FILENAME (TRUE | FALSE) "FALSE"

IS_KEY (TRUE | FALSE) "FALSE"

TRANSIENT (TRUE | FALSE) "FALSE"

COMPUTE_EXPR CDATA #IMPLIED

REPLACE_FETCHED_VALUE (TRUE | FALSE) "FALSE"

COLUMN_NAME CDATA #IMPLIED

IS_LONG_TEXT (TRUE | FALSE) "FALSE"

IS_DATE (TRUE | FALSE) "FALSE"

STATELESS_ALERTS (TRUE|FALSE) "FALSE"

IS_TIMESTAMP (TRUE | FALSE) "FALSE"

NON_THRESHOLDED_ALERTS (TRUE | FALSE) "FALSE"

KEYONLY_THRESHOLDS (TRUE | FALSE) "FALSE"

RENDERABLE (TRUE | FALSE) "TRUE"

HELP CDATA #IMPLIED>
```

## Attributes

NAME: This is the metric column name.

TYPE: Specifies the data type.

Supported types are:

a) NUMBER (default)

b) STRING.

c) RAW: This is for nested table support. A column may be defined as RAW to indicate it is a nested-table. Note: Only 1 level of nested table is allowed.

d) CLOB: CLOB holds large character data such as a log file.

e) BLOB: BLOB holds binary data (.zip files, .tar, files, etc.).

IS_KEY: is set to true if this column is the primary key (uniquely identifies the row in the returned rows). For any two rows returned, the value of the key column cannot be the same, else there will be a primary key violation. Note: Upto 5 columns can be marked with IS_KEY=TRUE. If no column is defined as key, the default value for the key is null (therefore should only return 1 row at a time)

TRUE | FALSE (default)

TRANSIENT: This will not be uploaded to repository. Only used to calculate rate data.

TRUE | FALSE (default)

COMPUTE_EXPR: This attribute specifies the formula for calculating the value of the column. Columns previously defined in the Table descriptor can participate in the calculation. Attaching a '_' prefix to a column name denotes previous value of a column. Refer to the Examples for details about the expression grammar and usage.

Predefined special values:

a) __interval: collect interval.

b) __sysdate: current system time.

c) __GMTdate: current GMT time.

d) __contains: tests a given string expression for presence of a string expression.

e) __beginswith: tests whether a given string expression begins with a specified string expression.

f) __endswith: testw whether a given string expression ends with the specified string expression.

g) __matches: tests whether a given string expression matches a specified string expression.

h) __delta: computes the difference between the current value and the previous value.

i) __leadingchars: returns the leading characters in the specified string.

j) __trailingchars: returns the trailing characters in the specified string.

k) __substringpos: returns the position of the occurrence of the pattern within a specified string.

l) __is_null: tests whether the expression is NULL

m) __length: returns the length of the string expression.

n) __to_upper: converts the string to upper case.

o) __to_lower: converst the string to lower_case.

p) __ceil: returns the smallest integral value not less than identifier.

q) __floor: returns the largest integral value not greater than the identifier.

r) __round: rounds to nearest integer, away from zero.

Prior to version 11, columns with COMPUTE_EXPR attributes could only be present after all the fetchlet returned columns. As part of bug 4869048, special compute expr columns will be allowed to be mixed in columns that get values from fetchlets.

COLUMN_NAME: This value will be used if the metric type is RAW to identify the database column.

IS_LONG_TEXT: This value will only be used when the metric is RAW and the column will be in digested form. The agent has support for metrics that expect to return the same long string repeatedly in metric results. If a column is marked with IS_LONG_TEXT="TRUE", the agent sends a row mapping the string to a digest into the MGMT_LONG_TEXT table and thereafter only sends the digested value as the data to the repository.

TRUE | FALSE (default)

IS_DATE: This value will only be used when the metric is RAW and the column is date type.

TRUE | FALSE (default)

STATELESS_ALERTS: If this attribute is set to TRUE, it indicates to Enterprise Manager that alerts on this column will not have corresponding clears. This allows the UI to decide whether to allow users to manually clear alerts on this column.

TRUE | FALSE (default)

IS_TIMESTAMP: The value in this column will be used as the collection time for this row. If set to true, the values for this column should be specified in the yyyy-MM-dd HH:mm:ss z format (For example: "2003-07-30 08:41:05 PST"). The list of valid time zones is listed in the $ORACLE_HOME/sysman/emd/supportedtzs.lst file.

HELP: Not used.

IS_FILENAME: When set to TRUE, it indicates that the column value is a file name that contains the real content that needs to be sent. IS_FILENAME attribute is valid only for CLOB/BLOB column types.

NON_THRESHOLDED_ALERTS: This attribute is used to indicate that there might be alerts for the metric column without there being a thresholded condition for it (eg: through server generated alerts).

TRUE | FALSE (default)

KEYONLY_THRESHOLDS: If this attribute is set to TRUE, conditions cannot apply to all metric rows and all Condition elements for the column need a KeyColumn element.

TRUE | FALSE (default)

RENDERABLE: A FALSE value for this attribute indicates that the value for this column maybe generated by the engine and may be cryptic or random enough to be of any use to the user. The UI would not display this value and would not allow the user to set thresholds for this value.

TRUE (default) | FALSE

REPLACE_FETCHED_VALUE: This attribute is only applicable if COMPUTE_EXPR is set, and for inner compute expressions (see above). If set, this tells the agent to use the compute expression value in place of the corresponding column returned by the fetchlet. If not set, the metric engine will right shift all existing fetchlet returned values to make place for the computed value.

TRUE | FALSE (default)

## Elements

Display

CategoryValue

TableDescriptor

## Used In

TableDescriptor

## Examples

Each column must specify the name and the data type for the column. The column can also be tagged as a key column. These column values qualify the value returned in the non-key columns.

For example: In a metric for top 10 processes, the process name will be the key column while the residence memory size, cpu, time used will be the value columns.

```
<ColumnDescriptor NAME="ciscoMemoryPoolName" TYPE="STRING" IS_KEY="TRUE">

<Display>

<Label NLSID="cisco_mem_pool_name">Memory Pool Name</Label>

</Display>

</ColumnDescriptor>
```

This is the most common usage of the ColumnDescriptor element. 'ciscoMemoryPoolName' is a key column in a metric. The values of this column are of type, 'STRING'. The optional Display element when included in the ColumnDescriptor, associates a UI Label with the Column.

```
<ColumnDescriptor NAME="pgScan" TYPE="NUMBER" IS_KEY="FALSE"
TRANSIENT="TRUE" HELP="NO_HELP"/>
```

The attribute, `TRANSIENT` indicates that the column is used for internal calculations only and should not be uploaded to the repository.

```
<ColumnDescriptor NAME="property" COLUMN_NAME="property" TYPE="STRING"
IS_KEY="TRUE" HELP="NO_HELP"/>
```

The `COLUMN_NAME` attribute will be used in RAW metrics to identify a database column.

```
<ColumnDescriptor NAME="log_file_message" TYPE="STRING" IS_KEY="FALSE"
IS_LONG_TEXT="TRUE"/>
```

`IS_LONG_TEXT` attribute, when set to TRUE, is an indication to the EMAgent to expect long values.

```
<ColumnDescriptor NAME="load_timestamp" COLUMN_NAME="load_timestamp"
TYPE="STRING" IS_DATE="TRUE" IS_KEY="FALSE"/>
```

This example defines a column 'load_timestamp' for a RAW metric with values in the date format.

```
<ColumnDescriptor NAME="log_file_match_count" TYPE="NUMBER" IS_KEY="FALSE"
STATELESS_ALERTS="TRUE"/>
```

This definition indicates that the alerts on the column, 'log_file_match_count' will not have corresponding clears. The UI can provide the users the option to manually clear alerts for this data.

The example for TableDescriptor describes a table metric with ColumnDescriptor elements.

> **Note:**
>
> It is invalid for a ColumnDescriptor to both be a key and a timestamp column.

CLOB/BLOB is only valid inside RAW metrics. When TYPE is set to CLOB or BLOB, the ColumnDescriptor can also have IS_FILENAME attribute set to TRUE, in which case, the column value is the name of the file whose content should be sent rather than the column value itself. For CLOB/BLOB columns, the destination columns in the repository table should also be of CLOB/BLOB type.

**Compute Expression Support**:

Supported Grammar:

```
expression := (cond_expr | (cond_expr ? cond_expr : cond_expr)
```

```
cond_expr := (string_expr |
```

```
(string_expr == string_expr) |
```

```
(string_expr < string_expr) |
```

```
(string_expr > string_expr) |
```

```
(string_expr <= string_expr) |
```

```
(string_expr >= string_expr) |
```

```
(string_expr __contains string_expr) |

(string_expr __beginswith string_expr) |

(string_expr __endswith string_expr) |

(string_expr __matches string_expr) |

(string_expr __delta string_expr))

string_expr := (simple_expr |

(simple_expr __leadingchars simple_expr) |

(simple_expr __trailingchars simple_expr) |

(simple_expr __substringpos simple_expr))

simple_expr := (term |

(simple_expr + term ) |

(simple_expr - term) )

term := (unary_expr |

(term * unary_expr ) |

(term / unary_expr ) )

unary_expr := (factor |

(__is_null factor) |

(__length factor) |

(__to_upper factor) |

(__to_lower factor) |

(__ceil factor) |

(__floor factor) |

(__round factor) )

factor := ( identifier |

string_literal |

number |

'(' expression ')' )

string_literal := '\'' (character | "\\'" )* '\''
```

Usage:

```
<ColumnDescriptor NAME="pgScan" TYPE="NUMBER" />

<ColumnDescriptor NAME="pgScanRate" TYPE="NUMBER" IS_KEY="FALSE"
COMPUTE_EXPR="(pgScan-_pgScan)/__interval"/>
```

The value of the column is calculated using the given compute expression. The value of the column is calculated using the present value of the 'pgScan' column, the previous value of the same column ('_pgScan') and the collect interval. Note: 'pgScan' column should be defined before any column can use its value in the COMPUTE_EXPR.

```
<ColumnDescriptor NAME="baseDir" TYPE="STRING" />

<ColumnDescriptor NAME="component" TYPE="STRING" COMPUTE_EXPR="'/
httpd'" />

<ColumnDescriptor NAME="full_path" TYPE="STRING" COMPUTE_EXPR="baseDir +
component + '/egs/log/' "/>
```

The value of the column, 'full_path' is "<baseDir>/httpd/egs/log/" where <baseDir> is the value of the column baseDir.

```
<ColumnDescriptor NAME="value1" TYPE="NUMBER" COMPUTE_EXPR="Col __contains
'ay'" />

<ColumnDescriptor NAME="value2" TYPE="NUMBER" COMPUTE_EXPR="Col
__beginswith 'Mon'" />

<ColumnDescriptor NAME="value3" TYPE="NUMBER" COMPUTE_EXPR="Col_ _endswith
'day'" />

<ColumnDescriptor NAME="value4" TYPE="NUMBER" COMPUTE_EXPR="Col __matches
'Sun*'" />
```

If the value of Col, in the above metric, is "Sunday", the outcome of the COMPUTE_EXPR will be as follows:

```
value1 = 1

value2 = 0

value3 = 1

value4 = 1

<TableDescriptor>

<ColumnDescriptor NAME="parse_str" TYPE="STRING" IS_KEY="TRUE" />

<ColumnDescriptor NAME="startpos" TYPE="NUMBER" COMPUTE_EXPR="parse_str
__substringpos '#D1'" />

<ColumnDescriptor NAME="num_trailing" TYPE="NUMBER"
COMPUTE_EXPR="(__length parse_str) - startpos" />

<ColumnDescriptor NAME="trim_str" TYPE="STRING"
COMPUTE_EXPR="parse_str__trailingchars num_trailing" />

<ColumnDescriptor NAME="endpos" TYPE="NUMBER" COMPUTE_EXPR="trim_str
__substringpos '#E1'" />

<ColumnDescriptor NAME="result_str" TYPE="STRING" COMPUTE_EXPR="trim_str
__leadingchars endpos" />

</TableDescriptor>
```

The sample TableDescriptor describes a simple method for extracting a substring from a given string. If the data represented by "parse_str" is of the form:

#A1 10

#B1 20

#C1 30

#D1 40

#E1 50

#F1 60

The "result_str" has the value "#D1=40".

# CategoryValue

A CategoryValue element indicates the category for a metric, column or condition under a particular classification. If a CategoryValue is defined for a Metric element, it is valid for all the ColumnDescriptors in that Metric. If it is defined for a ColumnDescriptor, that column will have a category value that overrides the union of what it has and what is defined for the Metric.

The following MetricCategories are predefined for the MetricClass, 'FUNCTIONAL':

a) FAULT: metrics that can be used to indicate a breakdown in a component or occurrence of an error that indicates some component or user is unable to successfully complete processing. Example: AlertLog – Archiver hung

b) WORKLOAD_VOLUME: metrics that capture the workload on a system induced in proportion to the user's or batch jobs running against the system. It usually is an indication of how much work is done. Example: User calls (per second)

c) WORKLOAD_TYPE: metrics that capture the type of workload on a system independent of demand. It usually is an indication of what kind of work is done. Example: Logical Reads (per transaction)

d) PERFORMANCE: metrics that can be classified to measure the performance of a system. It usually is an indication of how well the system is doing. Example: Database response (per second)

e) CAPACITY: metrics that measure the usage of a fixed resource. Example: CPU Usage (per second)

f) CONFIGURATION: metrics that check the configuration of a target against a recommended best-practice configuration.

g) SECURITY: metrics that relate to the security aspects of the system.

<!ELEMENT CategoryValue EMPTY>

<!ATTLIST CategoryValue

CLASS CDATA #REQUIRED

CATEGORY_NAME CDATA #REQUIRED>

The category class of a metric should be 'DEFAULT' by default. Following are the category names for the 'DEFAULT' class:

- Availability: Metric that collects target availability related data, for example, Response metric.

- BusinessKPI: Metric that measures KPI related information are included under this category.

- Capacity: Metric that measures the usage of a fixed resource, for example CPU Usage (per second).

- Fault: Metric that can be used to indicate a breakdown in a component or occurrence of an error. This error indicates that some component or user is unable to successfully complete a specified process, for example, AlertLog – Archiver hung.

- Load: Metric that captures the load on a system induced in proportion to the user's or batch jobs running against the system. It is usually an indication of how much work is done, for example, User calls (per second).

- LoadType: Metric that captures the type of load on a system independent of demand. It is usually an indication of what kind of work is done, for example, Logical Reads (per transaction).

- Security: Metric that relates to the security aspects of the system.

- Utilization: Metric that captures the utilization of resources or service.

- Response: Metric that can be classified to measure the response of a system. It is usually an indication of the system status (how well it is doing), for example, Database response (per second).

- Error: Metric that captures errors that are occurring are included in this category, for example, ORA error occurring in a database alert log.

## Attributes

CLASS: Name of the metric class.

CATEGORY_NAME: Name of the metric category.

## Elements

None.

## Used In

Metric

ColumnDescriptor

Condition

## Examples

```
<TargetMetadata TYPE="example3" META_VER="2.0">

<MetricClass NAME="EvaluationCost" NLSID="id_for_eval_cost_class">

<MetricCategory NAME="CHEAP" NLSID="id_for_cheap_cat"/>

<MetricCategory NAME="MEDIUM" NLSID="id_for_medium_cat"/>
```

```
<MetricCategory NAME="EXPENSIVE" NLSID="id_for_expensive_cat"/>

</MetricClass>

<Metric NAME="metric1" TYPE="TABLE">

<CategoryValue Class="EvaluationCost" CATEGORY_NAME="CHEAP"/>

. . .

</Metric>

. . .

</TargetMetadata>
```

This example illustrates the use of CategoryValue for a Metric. "metric1" is "CHEAP" to evaluate.

```
<Metric NAME="FileSystems" TYPE="TABLE" >

<CategoryValue CLASS="FUNCTIONAL" CATEGORY="WORKLOAD_VOLUME" />

<TableDescriptor>

<ColumnDescriptor NAME="FileSystem" TYPE="STRING" IS_KEY="TRUE" />

<ColumnDescriptor NAME="totalSpace" TYPE="NUMBER" >

<CategoryValue CLASS="FUNCTIONAL" CATEGORY="CAPACITY" />

</ColumnDescriptor>

<ColumnDescriptor NAME="diskUsedPct" TYPE="NUMBER />

<TableDescriptor>

....

</Metric>
```

In this sample, the column, "totalSpace" has a CategoryValue, "CAPACITY" which overrides the CategoryValue, "WORKLOAD_VOLUME" associated with the metric.

## CustomTableMapper

The CustomTableMapper element is part of the mid tier based versioning project that allows custom (RAW) metrics to change their destination tables based on the version of the mid tier.

The TableDescriptor for a RAW metric can have multiple CustomTableMapper elements - one per set of mid tier versions - with each CustomTableMapper providing repository table and column mappings for the TableDescriptor's Columns.

```
<!ELEMENT CustomTableMapper (ValidMidTierVersions*, ColumnMapper*)>

<!ATTLIST CustomTableMapper

REP_TABLE_NAME CDATA #REQUIRED>
```

## Attributes

REP_TABLE_NAME: Indicates the table name that the content of the metric should be uploaded to.

## Elements

ValidMidTierVersions

ColumnMapper

## Used In

TableDescriptor

CustomTableMapper

## Examples

```
<TableDescriptor>

<ColumnDescriptor NAME="c1" COLUMN_NAME="col1" TYPE="STRING"/>

<ColumnDescriptor NAME="c2" COLUMN_NAME="col2" TYPE="STRING"/>

<CustomTableMapper REP_TABLE_NAME="table1">

<ValidMidTierVersions PLUG_IN="DB" START_VER="10.1" END_VER="10.3"/>

<ColumnMapper METRIC_COLUMN="col1" REP_TABLE_COLUMN="col1_rep"/>

</CustomTableMapper>

</TableDescriptor>
```

This sample illustrates Mid-tier based versioning. TableDescriptor element must not contain the TABLE_NAME attribute. The sample describes a mapping of the metric to the repository table, 'table1' for Mid-tier versions between 10.1 (inclusive) and 10.3 (not inclusive). Metric column, 'col1' maps to 'col1_rep' in the repository table, 'table1'.

Refer to the example for ValidMidTierVersions also.

# ColumnMapper

The ColumnMapper element is part of a CustomTableMapper element and describes the mapping between the ColumnDescriptor and the repository column its data should end up in.

The presence of a ColumnMapper provides the mapping for the column in a particular repository table, and indicates that the column is required in the table. To indicate that a column should not be uploaded to a particular version of the repository, there should not be a ColumnMapper for that column

```
<!ELEMENT ColumnMapper EMPTY>

<!ATTLIST ColumnMapper
```

```
METRIC_COLUMN CDATA #REQUIRED

REP_TABLE_COLUMN CDATA #REQUIRED>
```

## Attributes

METRIC_COLUMN: Name of the ColumnDescriptor this applies to

REP_TABLE_COLUMN: The database table column name this data should end up in.

## Elements

None.

## Used In

CustomTableMapper

## Examples

Refer to the Examples for CustomTableMapper.

# QueryDescriptor

The query descriptor allows the framework to find the fetchlet as well as pass on the query information for obtaining performance data values from the target. The fetchlet can be identified by a well known id that is known to the EMAgent. It may also contain properties that will be passed to the fetchlet.

```
<!ELEMENT QueryDescriptor (ValidIf*, Property*) >

<!ATTLIST QueryDescriptor

FETCHLET_ID CDATA #REQUIRED

NEED_CHARSET_CONVERT (TRUE | FALSE) "TRUE"

REMOTE (TRUE | FALSE) "FALSE"

ON_TARGET (TRUE | FALSE) "FALSE"

>
```

## Attributes

FETCHLET_ID: Specifies the ID of the fetchlet to use that is known to the EMAgent. This attribute must point to an element from the $ORACLE_HOME/lib/fetchlets.reg file.

NEED_CHARSET_CONVERT: If the metric result is in correct "UTF8" encoding, this flag should be set to "FALSE" so that EMAgent will not do any character conversion.

TRUE (default) | FALSE

REMOTE: This attribute enables the agent to select the appropriate QueryDescriptor based on whether the target is local or remote. A single QueryDescriptor with REMOTE=false indicates that the same querydescriptor will be used for local and remote targets. A single QueryDescriptor with REMOTE=true indicates that the

querydescriptor will be used ONLY if the target is remote and skipped if the target is remote. If there are 2 QueryDescriptors one with REMOTE=true, the other MUST be REMOTE=false.

TRUE | FALSE (default)

ON_TARGET: A value of TRUE indicates that the metric must be evaluated on the target whereever it may be. For a local target, this attribute is ignored. For a remote target, the metric is evaluated over SSH.

TRUE | FALSE (default)

## Elements

ValidIf

Property

## Used In

Metric

DynamicProperties

## Examples

The query descriptor associated will contain the metadata that can be used to collect the value of the metric. For example: SQL Query.

```
<TargetMetadata TYPE="example1" META_VER="2.0">

<Metric NAME="prop" TYPE="TABLE">

<TableDescriptor>

<ColumnDescriptor NAME="name" TYPE="STRING" IS_KEY="TRUE" />

<ColumnDescriptor NAME="value" TYPE="STRING" />

</TableDescriptor>

<QueryDescriptor FETCHLET_ID="OS">

<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>

</QueryDescriptor>

</Metric>

</TargetMetadata>
```

This simple example describes a query descriptor that is used in a Metric and relies on the 'OS' fetchlet to return the property 'hostname'.

```
<TargetMetadata TYPE="example1" META_VER="2.0">

<Metric NAME="prop" TYPE="TABLE">

. . .

</Metric>
```

```
<InstanceProperties>

<DynamicProperties NAME="VersionAndLocation" FORMAT="ROW"
PROP_LIST="OS;OracleHome;Version">

<QueryDescriptor FETCHLET_ID="OSLineToken">

<Property NAME="ENVEmdOS" SCOPE="SYSTEMGLOBAL">_emdOS</Property>

<Property NAME="ENVVersion" SCOPE="SYSTEMGLOBAL">_emdVersion</Property>

<Property NAME="ENVORACLE_HOME" SCOPE="SYSTEMGLOBAL">emdRoot</Property>

</QueryDescriptor>

</DynamicProperties>

</InstanceProperties>

</TargetMetadata>
```

This example illustrates the use of a QueryDescriptor to evaluate a DynamicProperties element.

## Property

Describes the information to be passed to the fetchlets.

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

<!ATTLIST Property

NAME CDATA #REQUIRED

SCOPE (GLOBAL | INSTANCE | USER | SYSTEMGLOBAL | ENV | HOST | CACHE)
"GLOBAL"

OPTIONAL (TRUE | FALSE) "FALSE">
```

Property values are resolved as follows:

1. The value is looked up in the specified scope

2. For each potential instantiation (ie %<varname>%) in the looked up value, varname is looked up as follows:

a. In the property values itself (ie in one of the earlier properties).

b. In the instance properties

c. In the systemglobal scope (emd.properties)

d. The value is checked for automatic property.

The following Automatic Properties are defined for a target:

1. NAME - substitutes Target Name

2. TYPE - substitutes Target type

3. DISPLAY_NAME – substitutes display name for the target

4. TYPE_DISPLAY_NAME – substitutes display name for the type

5. GUID – substitutes the guid

> ✏️ **Note:**
>
> All lookups are case-sensitive.

## Attributes

NAME: Name of the property.

SCOPE: Defines how the value of the property is to be resolved.

Supported values for Scope are:

a) GLOBAL (default): The property needs to be resolved in the Target Type Definition XML file.

b) INSTANCE: The property will be resolved by discovery. The PCDATA in that case should be the NAME of the property set in the discovery XML file.

c) USER: The property will be resolved by the caller(collector or the interactive end-user). The PCDATA in that case should be the name of the Property to be used when prompting the caller (in the case of interactive user).

d) SYSTEMGLOBAL: Use emd.properties to resolve the property.

e) ENV: Use environment variable to resolve the property.

f) HOST: The property must be resolved as an instance variable of the 'host' target on that EMAgent. For example, the OS property

g) CACHE: The value must be obtained from the previous evaluation of the metric. Any column returned in the previous evaluation can be specified, and this is only applicable to single row OR non-key metrics.

OPTIONAL: is meant to call out those properties that need NOT be available when provided to the fetchlet. The EMAgent will validate that it can find valid values for all non-optional properties before calling through to a fetchlet.

TRUE | FALSE (default)

## Elements

Contains character data representing the value for the property.

## Used In

QueryDescriptor

PushDescription

## Examples

```
<Property NAME="perlBin" SCOPE="SYSTEMGLOBAL">perlBin</Property>
```

'perlBin' Property has 'SYSTEMGLOBAL' scope which implies that emd.properties file is used to resolve the property

```
<Property NAME="delimiter" SCOPE="GLOBAL">|</Property>
```

'delimiter' property has 'GLOBAL' scope.

```
<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>
```

'hostname' has 'INSTANCE' scope which implies that the value will be resolved by discovery. The value 'NAME' must match the field in the discovery XML file.

```
<Property NAME="SNAPSHOT_TYPE" SCOPE="USER">SNAPSHOT_TYPE</Property>
```

This property will be resolved by the caller. 'SNAPSHOT_TYPE' is the name of the property when prompting the caller.

```
<Property NAME="ENVNMUPM_TIMEOUT" OPTIONAL="TRUE"
SCOPE="SYSTEMGLOBAL">NMUPM_TIMEOUT</Property>
```

The property, 'ENVNMUPM_TIMEOUT' is identified as an OPTIONAL property. All properties that are not OPTIONAL have to be validated by the EMAgent before the fetchlet is called.

Use of the property elements is illustrated in the following examples:

QueryDescriptor, ValidIf, TargetMetadata.

# Label

This represents the Label that will be displayed in UI.

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

```
<!ATTLIST Label
```

```
NLSID CDATA #REQUIRED>
```

## Attributes

NLSID: Will contain the ID used to lookup the string in to the resource bundle

## Elements

Character data.

## Used In

[Display](#)

## Examples

This element must be present in the Display element.

```
<Label NLSID="host_load_cpuLoad">Run Queue Length (5 minute average)</
Label>
```

This element is defined in a Display element and represents the label to display.

Refer to the example for Display.

# ShortName

ShortName is the short representation of the Metric Display name it should be less than 12 characters in length.

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

<!ATTLIST ShortName

NLSID CDATA #REQUIRED>
```

## Attributes

NLSID: Will contain the ID used to lookup the string in to the resource bundle

## Elements

Character data.

## Used In

Display

## Examples

```
<ShortName NLSID="host_load_cpuLoad_short">CPU Load (5min)</ShortName>
```

This element is defined in a Display element and represents the short name to display.

Refer to the example for Display.

# Icon

This represents the icon to be used.

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

<!ATTLIST Icon

GIF CDATA #IMPLIED>
```

## Attributes

GIF: Will contain the name and the filepath for the image to be displayed.

## Elements

Character data.

## Used In

Display

## Examples

Refer to the example for Display.

## Description

This holds the description of the displayed entity.

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

<!ATTLIST Description

NLSID CDATA #IMPLIED>
```

> **Note:**
>
> It is preferred to have a description of the metric, for example what information it is collecting. If the description is provided, then it is displayed on the **All metrics** page.

## Attributes

NLSID: Will contain the ID used to lookup the string in to the resource bundle.

## Elements

Character data.

## Used In

Display

## Examples

Refer to the example for Display.

## Unit

This holds the Unit information for the displayed data.

There are some standard units and unit nls ids that are supported. Use the appropriate nls ids and display names for these standard units mentioned below. The translation for these system supported units (nls ids that start with "em__sys__"), is done at the system level and does not need to be translated on a per target type basis.

Supported Units:

Standard Percent: used for metrics who values are between 0 and 100%

NLSID: "em__sys__standard_percent"

Display: "%"

Usage: <Unit NLSID="em__sys__standard_percent">%</Unit>

Generic Percent: used for metrics who values can be in +ve and -ve percentages as well - like -50% or 200%

NLSID: "em__sys__generic_percent"

Display: "%"

Usage: <Unit NLSID="em__sys__generic_percent">%</Unit>

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

```
<!ATTLIST Unit
```

```
NLSID CDATA #IMPLIED>
```

## Attributes

NLSID: Will contain the ID used to lookup the string in to the resource bundle.

## Elements

Character data.

## Used In

Display

## Examples

Refer to the example for Display.

# MonitoringMode

MonitoringMode element indicates the mediator for data collection. Presence of this element in TargetMetadata element, indicates that the target is of cluster type. Mediation is required for a cluster type target to provide data collection consistency across all cluster type target agents. Cluster targets can be OMS mediated, Agent mediated or Repository mediated. MEDIATOR attribute specifies the mediation. CLUSTERDESCRIPTOR attribute points to the shared library that implements the cluster interfaces needed by the agent. This is applicable only for AgentMediated clusters.

```
<!ELEMENT MonitoringMode (ValidIf*)>
```

```
<!ATTLIST MonitoringMode
```

```
MEDIATOR (AgentMediated|OMSMediated) #REQUIRED
```

```
CLUSTERDESCRIPTOR CDATA #IMPLIED>
```

## Attributes

MEDIATOR: Specifies the mediator to use.

Supported values are:

a) AgentMediated

b) OMSMediated

CLUSTERDESCRIPTOR: Describes the type of the cluster. This is applicable for Agent mediation only.

## Elements

ValidIf

## Used In

TargetMetadata

## Examples

```
<TargetMetadata META_VER="2.0" TYPE="example1"
CATEGORY_PROPERTIES="OS;OSVersion">

. . .

<MonitoringMode MEDIATOR="OMSMediated">

<ValidIf>

<CategoryProp NAME="OSVersion" CHOICES="5.8"/>

<CategoryProp NAME="OSVersion" CHOICES="5.9"/>

</ValidIf

</MonitoringMode>

. . .

</TargetMetadata>
```

This example indicates the target, 'example1' is of cluster type and is OMS Mediated only for the OSVersions 5.8 and 5.9. For the other versions it acts like a normal target. All the Management Agents would monitor the target. Absence of this element makes the target a normal target.

# AltSkipCondition

The agent has logic to skip evaluation of metrics for targets that are known to be down to reduce generation of metric errors due to connection failures. Metrics are skipped whenever there is an error in evaluating the Response metric or there is a non-clear severity on the Response, Status condition. If a target needs to have its metric evaluation stop on a condition other than the Response, Status column, this can be specified by creating an AltSkipCondition element.

```
<!ELEMENT AltSkipCondition EMPTY >

<!ATTLIST AltSkipCondition

METRIC CDATA #REQUIRED

COLUMN CDATA #REQUIRED
```

```
ASSOC_TARGET CDATA #IMPLIED>
```

## Attributes

METRIC: Name of the result metric

COLUMN: Name of the column

ASSOC_TARGET: may be used to point to the conditions in an associated target.

## Elements

None.

## Used In

TargetMetadata

## Examples

```
<TargetMetadata TYPE="example1" META_VER="2.0">

<AltSkipCondition METRIC="metric1" COLUMN="Status"/>

<Metric NAME="Response" TYPE="TABLE">

<TableDescriptor>

<ColumnDescriptor NAME="Status" TYPE="NUMBER" IS_KEY="FALSE"/>

</TableDescriptor>

<QueryDescriptor FETCHLET_ID="OS">

<Property NAME="hostname" SCOPE="INSTANCE">NAME</Property>

</QueryDescriptor>

</Metric>

<Metric NAME="metric1" TYPE="TABLE">

<TableDescriptor>

<ColumnDescriptor NAME="Status" TYPE="NUMBER" IS_KEY="FALSE"/>

</TableDescriptor>

<QueryDescriptor FETCHLET_ID="OS">

<Property NAME="homedir" SCOPE="INSTANCE">home</Property>

</QueryDescriptor>

</Metric>

</TargetMetadata>
```

This example describes a target whose metric evaluation will be skipped whenever there is an error in evaluating the Response metric or there is a non-clear severity

on the Response Status column. In addition to that, metric evaluation will be skipped if errors or non-clear severity are encountered on the Status column of the 'metric1' metric as well.

```
<AltSkipCondition METRIC="Response" COLUMN="State" ASSOC_TARGET="t2"/>
```

If the 'example1' target type in the above example was associated with another target type. The AltSkipCondition can be used to skip evaluating example1's metric based on Response metric's 'State' column in the other target type.

The association 't2' has to be defined in the targets.xml file for example1.

# CredentialInfo

Credential types are metadata for sets of credentials. They describes the components of the credential (CredentialTypeColumns), which is the key etc. In some cases, CredentialTypes may be composed of existing CredentialTypes (in this or other target types)

CredentialSets are the instances of CredentialTypes that apply to a particular target. Of particular importance are the monitoring credential sets whose values are mapped to the instance properties of the target.

```
<!ELEMENT CredentialInfo (CredentialType*, CredentialSet*)>
```

## Attributes

None.

## Elements

CredentialType

CredentialSet

## Used In

TargetMetadata

## Examples

```
<TargetMetadata TYPE="example1" META_VER="2.0">

. . .

<CredentialInfo>

<CredentialType NAME="DBCreds" >

<CredentialTypeColumn NAME="DBUsername" IS_KEY="true" />

<CredentialTypeColumn NAME="DBPassword" />

<CredentialTypeColumn NAME="DBRole" />

</CredentialType>
```

```
<CredentialSet NAME="DBCredsMonitoring" CREDENTIAL_TYPE="DBCreds"
USAGE="monitoring">

<CredentialSetColumn TYPE_COLUMN="DBUsername" SET_COLUMN="UserName" />

<CredentialSetColumn TYPE_COLUMN="DBPassword" SET_COLUMN="password" />

<CredentialSetColumn TYPE_COLUMN="DBRole" SET_COLUMN="role" />

</CredentialSet>

<CredentialSet NAME="DBCredsSysdba" CREDENTIAL_TYPE="DBCreds"
USAGE="monitoring">

<CredentialSetColumn TYPE_COLUMN="DBUsername"
SET_COLUMN="SYSDBAUserName" />

<CredentialSetColumn TYPE_COLUMN="DBPassword"
SET_COLUMN="SYSDBApassword" />

<CredentialSetColumn TYPE_COLUMN="DBRole" SET_COLUMN="SYSDBArole" />

</CredentialSet>

</CredentialInfo>

. . .

</TargetMetadata>
```

CredentialInfo may have CredentialType and CredentialSet elements. This is illustrated in this example. Target type, 'example1' is associated with 'DBCreds', 'DBCredsMonitoring' and 'DBCredsSysdba' credentials.

# CredentialType

CredentialType elements contain the description of a type as composed of component columns (one of which may be the key) or as a composite of other predefined credential types.

```
<!ELEMENT CredentialType (Display?, (CredentialTypeColumn|
CredentialTypeRef)+)>

<!ATTLIST CredentialType

NAME CDATA #REQUIRED>
```

## Attributes

NAME: Unique Name of the CredentialType

## Elements

Display

CredentialTypeColumn

CredentialTypeRef

## Used In

CredentialInfo

## Examples

CredentialType may contain an optional Display element specifying the display characteristics for the CrendentialType and one or more of either the CredentialTypeColumn or the CredentialTypeRef.

```
<CredentialType NAME="HostCreds" >

<CredentialTypeColumn NAME="HostUserName" IS_KEY="TRUE">

<CredentialTypeColumn NAME="HostPassword">

</CredentialType>
```

In this sample 'HostCreds' is declared as a CredentialType. Refer to a more detailed example in CredentialInfo.

# CredentialTypeColumn

CredentialType is defined as a set of Credential Columns. Each CredentialTypeColumn may provide a list of values that are the only allowed values for this column.

```
<!ELEMENT CredentialTypeColumn (Display?, CredentialTypeColumnValue*)>

<!ATTLIST CredentialTypeColumn

NAME CDATA #REQUIRED

IS_KEY (TRUE|FALSE) "FALSE">
```

## Attributes

NAME: Name of the column

IS_KEY: If multiple sets are created of this credential type, is this the column by which one set is differentiated from another.

TRUE | FALSE (default)

## Elements

Display

CredentialTypeColumnValue

## Used In

CredentialType

## Examples

CredentialTypeColumn may contain an optional Display element specifying the display characteristics for the CredentialTypeColumn and optional CredentialTypeColumnValue element(s).

```
<CredentialTypeColumn NAME="HostUserName" IS_KEY="TRUE">

<Display FOR_SUMMARY_UI="TRUE">

<Label NLSID="host_username">UserName</Label>

</Display>

</CredentialTypeColumn>
```

The HostUserName column is the key in a (username,password) credential type. This column would be displayed in a condensed version of the UI and the label associated with this is 'UserName'. Refer to a more detailed example in CredentialInfo.

# CredentialTypeColumnValue

CredentialTypeColumnValue holds the allowed values for a CredentialTypeColumn.

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

<!ATTLIST CredentialTypeColumnValue

IS_DEFAULT (TRUE|FALSE) "FALSE">
```

## Attributes

IS_DEFAULT: Is set to true if this is the default value in the list.

TRUE | FALSE (default)

## Elements

Character data representing the value

## Used In

CredentialTypeColumn

## Examples

```
<CredentialType NAME="DBCreds" >

<CredentialTypeColumn NAME="DBUsername" IS_KEY="true" />

<CredentialTypeColumn NAME="DBPassword" />

<CredentialTypeColumn NAME="DBRole">

<CredentialTypeColumnValue IS_DEFAULT="true">normal</
CredentialTypeColumnValue>
```

```
<CredentialTypeColumnValue>sysdba</CredentialTypeColumnValue>
```

```
</CredentialTypeColumn>
```

```
</CredentialType>
```

The 'DBRole' column in 'DBCreds' credential type may have the following values:

1. normal (default value)

2. sysdba

Refer to the example in CredentialInfo. The example shows the context for the CredentialType element

# CredentialTypeRef

This element allows a credential type to refer to other predefined credential types. It contains mapping of the columns in the original credential type to columns of the credential type being defined.

```
<!ELEMENT CredentialTypeRef (CredentialTypeRefColumn*)>
<!ATTLIST CredentialTypeRef
REF_NAME        CDATA #REQUIRED
REF_TYPE        CDATA #REQUIRED
REF_TARGETTYPE  CDATA #IMPLIED
ASSOCIATION     CDATA #IMPLIED>
```

## Attributes

REF_NAME: Specifies the name for this `CredentialTypeRef`.

REF_TYPE: Credential type referred to.

REF_TARGETTYPE: The target type that contains the original credential type. Specify Null if this is the same target type.

ASSOCIATION: Refers to the association of this target with the other target for whom credentials are maintained here. Note that this value needs to be one of the AssocTarget elements above.

## Elements

CredentialTypeRefColumn

## Used In

CredentialType

## Examples

```
<CredentialType NAME="FMCreds" >
<CredentialTypeRef NAME="FMDBCreds1" REF_TYPE="DBCreds"
REF_TARGET_TYPE="oracle_database" ASSOCIATION="firstDB">
<CredentialTypeRefColumn NAME="FMUserName1" REF_TYPECOLUMN="DBUsername" />
<CredentialTypeRefColumn NAME="FMpassword1" REF_TYPECOLUMN="DBPassword" />
<CredentialTypeRefColumn NAME="FMRole1" REF_TYPECOLUMN="DBRole" />
```

**ORACLE**

```
                            </CredentialTypeRef>
        </CredentialType>
```

FMCreds defines a credential type whose columns FMUserName1, Fmpassword1 and FMRole are mapped to DBCred's DBUsername, DBPassword and DBRole columns respectively.

Refer to the example in CredentialInfo. The example shows the context for the CredentialType element

# CredentialTypeRefColumn

This element maps the columns in the referred credential type to this credential type's columns.

```
<!ELEMENT CredentialTypeRefColumn EMPTY>
<!ATTLIST CredentialTypeRefColumn
          NAME            CDATA #REQUIRED
          REF_TYPECOLUMN  CDATA #REQUIRED>
```

## Attributes

NAME: Name of column in this credential type.

REF_TYPECOLUMN: Name of the column in the referred credential type

## Elements

None

## Used In

CredentialTypeRef

## Examples

Refer to the examples for CredentialTypeRef.

# CredentialSet

This element defines a set of elements that form a named credential set for this target type. A credential set provides values for one of the credential types defined for this target type. The credential set may contain credentials for one of 3 usages: monitoring, preferred credentials or app specific functionality.

```
<!ELEMENT CredentialSet (Display?, CredentialSetColumn+)>
<!ATTLIST CredentialSet
          NAME             CDATA  #REQUIRED
          CREDENTIAL_TYPE  CDATA  #REQUIRED
          USAGE    (MONITORING|PREFERRED_CRED|SYSTEM) "MONITORING"
          CONTEXT_TYPE     (TARGET|CONTAINER|COLLECTION) "TARGET"
          CONTEXT          CDATA  #IMPLIED>
```

## Attributes

NAME: Name of the credential set

CREDENTIAL_TYPE: The credential type that this set provides values for.

USAGE: Is this credential set used for monitoring, as preferred credentials or for app specific stuff?Supported values are:

- MONITORING (default): Specifies credentials that management applications can use to connect directly to the target.

- PREFERRED_CRED: Specifies a user's preferred credentials.

- SYSTEM: Specifies a fixed set of credentials that are used by certain specialized applications (patching, cloning etc.)

- CONTEXT_TYPE: Specifies what kind of entity, the set pertains to.

  Supported values are:

  - TARGET (default): These are the stored credentials for a target that could be used by applications such as job system, patch etc.

  - CONTAINER: These are the stored credentials for a container. These are always host credentials.

  - COLLECTION: These are credentials associated with user-defined metrics.

- CONTEXT: Specifies the metric that this set is for. Refers to collection credentials only.

## Elements

Display

CredentialSetColumn

## Used In

CredentialInfo

## Examples

CredentialSet contains an optional Display element that specifies the display characteristics for the CredentialSet element and at least 1 CredentialSetColumn.

```
<CredentialSet NAME="HostPrefCreds" CREDENTIAL_TYPE="HostCreds"
USAGE="PREFERRED_CRED">
     <CredentialSetColumn TYPE_COLUMN="HostUsername"
SET_COLUMN="HostPrefUserName" />\
     <CredentialSetColumn TYPE_COLUMN="HostPassword"
SET_COLUMN="HostPrefPassword" />
</CredentialSet>
```

This sample illustrates usage of a user credential set.

```
<CredentialSet NAME="DBCredsMonitoring" CREDENTIAL_TYPE="DBCreds"
USAGE="monitoring">
     <CredentialSetColumn TYPE_COLUMN="DBUsername" SET_COLUMN="UserName"/>
     <CredentialSetColumn TYPE_COLUMN="DBPassword" SET_COLUMN="password"/>
     <CredentialSetColumn TYPE_COLUMN="DBRole" SET_COLUMN="role"/>
</CredentialSet>
```

ORACLE®

`DBCredsMonitoring` defines a credential set that may be used by the agent for monitoring a database.

```
<CredentialSet NAME="HostSystemCreds" CREDENTIAL_TYPE="HostCreds" USAGE="SYSTEM">
    <CredentialSetColumn TYPE_COLUMN="HostUsername"
SET_COLUMN="HostPrefUserName"/>
    <CredentialSetColumn TYPE_COLUMN="HostPassword"
SET_COLUMN="HostPrefPassword"/>
</CredentialSet>
```

`HostSystemCreds` is an example of a System credential type.

Refer to the example in CredentialInfo. The example shows the context for the CredentialType element.

# CredentialSetColumn

Credential set columns map the columns of a credential type to the source of their values. In the case of monitoring credential sets, the source is instance properties of the target.

```
<!ELEMENT CredentialSetColumn (Display?, CredentialSetColumnValue*) >
<!ATTLIST CredentialSetColumn
          TYPE_COLUMN CDATA #REQUIRED
          SET_COLUMN  CDATA #REQUIRED>
```

## Attributes

TYPE_COLUMN: Name of the column in the CredentialType.

SET_COLUMN

## Elements

Display

CredentialSetColumnValue

## Used In

CredentialSet

## Examples

`CredentialSetColumn` contains an optional Display element that specifies the display characteristics for the `CredentialSetColumn` element and optional `CredentialSetColumnValue` element(s).

Refer to the example for  CredentialSet

# CredentialSetColumnValue

This element holds the allowed values for a CredentialSetColumn.

```
<!ELEMENT CredentialSetColumnValue (#PCDATA)>
<!ATTLIST CredentialSetColumnValue
          IS_DEFAULT (TRUE|FALSE) "FALSE">
```

## Attributes

IS_DEFAULT: Set this attribute to true if the element represents the default value in the list.

## Elements

Character data representing the value.

## Used In

CredentialSetColumn

## Examples

Refer to the example for CredentialSet.

# InstanceProperties

The InstanceProperties element declares the "properties" of a target type. Some properties are obtained from the targets.xml file, and may be optional or required, and others can be computed using DynamicProperties elements using the values of other properties.

The Agent uses the information in the InstanceProperties element to determine when a target has not been sufficiently configured, and to compute the dynamic properties for the target. The console UIs use the information about the properties to create UIs where a target can be created from scratch or an existing target's properties are modified.

InstanceProperties holds target InstanceProperty(s) and DynamicProperties elements.

```
<!ELEMENT InstanceProperties ((InstanceProperty | DynamicProperties)*)>
```

## Attributes

None

## Elements

InstanceProperty

DynamicProperties

## Used In

TargetMetadata

## Examples

Refer to the example for QueryDescriptor and TargetMetadata.

# InstanceProperty

An `InstanceProperty` element contains the definition of an instance property.

```
<!ELEMENT InstanceProperty (ValidIf*, (PCDATA | Display)*)>
<!ATTLIST InstanceProperty
NAME CDATA #REQUIRED
OPTIONAL (TRUE | FALSE) "FALSE"
IN_PRIMARY_KEY (TRUE | FALSE) "FALSE"
CHECK_FOR_MODIFIABLE (TRUE | FALSE) "FALSE"
CREDENTIAL (TRUE | FALSE) "FALSE"
READONLY (TRUE | FALSE) "FALSE"
NEED_REENTER ( TRUE | FALSE) "FALSE"
HIDE_ENTRY ( TRUE | FALSE) "TRUE"
CHECK_ORIGINAL ( TRUE | FALSE) "FALSE"
IS_COMPUTED (TRUE|FALSE) "FALSE"
WAS_REQUIRED (FALSE|TRUE) "FALSE"
>
```

## Attributes

NAME: Name of the property

OPTIONAL: Is a value for the property required

TRUE | FALSE (default)

CREDENTIAL: Is the property sensitive in nature. Such properties are usually saved obfuscated in targets.xml

TRUE | FALSE (default)

READONLY: Marks this element as ReadOnly.

TRUE | FALSE (default)

NEED_REENTER: If TRUE, will require user to enter the value twice at command line.

TRUE | FALSE (default)

HIDE_ENTRY: If TRUE, will show the character user typed as '*'.

TRUE (default) | FALSE

CHECK_ORIGINAL: If TRUE, before modify, user need to type the original value.

IS_COMPUTED: If TRUE, indicates that it describes a dynamic property.

WAS_REQUIRED: This was an instance property that was required prior to 10.2 but is now a dynamic property. This property would be sent to 10.1 OMS if set to TRUE. Default value is FALSE (See bug/ER 4631553 for more details).

## Elements

ValidIf

Display

## Used In

TargetMetadata

## Examples

If InstanceProperty element contains ValidIf elements, all the conditions must be met for the property to be evaluated. InstanceProperty also optionally contains either Display elements or character data.

```
<InstanceProperty NAME="password" OPTIONAL="FALSE" CREDENTIAL="TRUE">
```

Example: for an oracle_database target one InstanceProperty has the NAME "password", which is not OPTIONAL and which is a "CREDENTIAL".

Refer to additional examples for InstanceProperty described in TargetMetadata

# DynamicProperties

DynamicProperties elements allow a target to specify a query that will return a set of values corresponding to the instance properties of the target. The values are turned into target properties and are accessible to other query descriptors from the INSTANCE scope.

```
<!ELEMENT DynamicProperties (ValidIf*, (QueryDescriptor | ExecutionDescriptor)+)
>
<!ATTLIST DynamicProperties
NAME CDATA #REQUIRED
PROP_LIST CDATA #IMPLIED
OPT_PROP_LIST CDATA #IMPLIED
FORMAT (TABLE | ROW) "TABLE"
IS_CRITICAL (TRUE | FALSE) "FALSE"
>
```

> **Note:**
>
> If `CategoryProperties` are instantiated through `DynamicProperty` evaluation, such a failed `DynamicProperty` evaluation would cause the agent to reject the target unless the value of the category property is available in targets.xml.

## Attributes

NAME: attribute simply identifies the property collector, for error tracing etc.

PROP_LIST: Contains ';' separated values that specify a list of names of properties that can be returned by the query descriptor. The result MUST contain the properties listed here.

OPT_PROP_LIST: Contains ';' separated values that specify a list of names of properties that can be returned by the query descriptor. The result MAY contain the properties listed here.

FORMAT: Specifies the format for the return data.

Supported values are:

- TABLE (default): If the FORMAT is "TABLE" (the default), the return value must be a table of instance property values. The table returned must be a two-column (NAME, VALUE) table.

- ROW: If the FORMAT is "ROW", the contents of the one row are taken as the values of the properties in the same order they are listed in the PROP_LIST, and then in the OPT_PROP_LIST lists.

IS_CRITICAL: To denote that a dynamic property is cirtical for the target one can specify the IS_CRITICAL flag attribute. The default value of this flag is FALSE and is assumed if this attribute is not present. When the value of this attribute is specified as TRUE, then on failure / timeout during computation of that dynamic property, we reschedule the computation of all dynamic / instance properties of that target as per the values of parameters given in the emd.properties file.

## Elements

ValidIf

QueryDescriptor

ExecutionDescriptor

## Used In

InstanceProperties

## Examples

If `DynamicProperties` element contains `ValidIf` elements, all the conditions must be met for the property to be evaluated. In addition to this it must also include at least 1 instance of either `QueryDescriptor` or `ExecutionDescriptor`.

The property names in the `PROP_LIST` and `OPT_PROP_LIST` are used in conjunction with `InstanceProperty` declarations while validating target type metadata.

```
<DynamicProperties NAME="VersionAndLocation" FORMAT="ROW"
PROP_LIST="OS;OracleHome;Version">
```

`DynamicProperties`, 'VersionAndLocation' has a ROW format and returns 'OS', 'OracleHome', 'Version' as properties.

Note that if a query descriptor returns a property value that is already available, the property is ignored. If multiple `DynamicProperties` queries return a property, the value from the first one is used.

The example in TargetMetadata includes the context for DynamicProperties element.

# ExecutionDescriptor

ExecutionDescriptor specifies the execution plan for evaluating a metric. MAgent executes each statement of the plan, in the order it is defined, to produce a Metric

Result. The Metric Result generated as result of the evaluation of the last statement of the execution plan will be returned.

```
<!ELEMENT ExecutionDescriptor (ValidIf*, (GetTable | GetView | GroupBy | Union |
JoinTables)*)>
```

## Attributes

None

## Elements

ValidIf

GetTable

GetView

GroupBy

Union

JoinTables

## Used In

Metric

DynamicProperties

## Examples

If `ExecutionDescriptor` element contains `ValidIf` elements, all the conditions must be met for the element to be evaluated. In addition to this it must also include 0 or more instances of either 1 of the following elements: `GetTable`, `GetView`, `GroupBy`, `Union`, `JoinTables`

`ExecutionDescriptor` is used to compute aggregation metric.

```
<TargetMetadata META_VER="3.0" TYPE="host">
. . .
<Metric NAME="Load" TYPE="TABLE">
. . .

<ExecutionDescriptor>
<GetTable NAME="DiskActivity"/>
<GetView NAME="AvgSrvcTimeView" FROM_TABLE="DiskActivity">
   <Column NAME="DiskActivityavserv"/>
</GetView>
<GroupBy NAME="DA_MaxAvServ" FROM_TABLE="AvgSrvcTimeView">
<AggregateColumn NAME="longestServ" COLUMN_NAME="DiskActivityavserv"
OPERATOR="MAX" />
</GroupBy>
<GetTable NAME="_LoadInternal"/>
<JoinTables NAME="Load">
<Table NAME="_LoadInternal"/>
<Table NAME="DA_MaxAvServ"/>
</JoinTables>
</ExecutionDescriptor>
```

```
</Metric>

<Metric NAME="_LoadInternal" TYPE="TABLE" USAGE_TYPE="HIDDEN">
   . . .
</Metric>

<Metric NAME="DiskActivity" TYPE="TABLE">
   . . .
</Metric>

                   </TargetMetadata>
```

```
This ExecutionDescriptor in this sample generates the following intermediate
Metric results after executing each statement:
§        GetTable: The metric result, 'DiskActivity' contains all the columns
of 'DiskActivity' metric.  The result of this operation is similar to the SQL
statement,
"Select * from DiskActivity"
§        GetView: The metric result, 'AvgSrvcTimeView' contains only
'DiskActivityavserv' column of 'DiskActivity' metric.
§        GroupBy: Metric result, 'DA_MaxAvServ' is a grouping of the
'AvgSrvcTimeView' based on the 'longestServ' which is the max value of the
'DiskActivityavserv' column.
§        GetTable: Metric result, '_LoadInternal' contains all the columns of
the '_LoadInternal' metric.
§        JoinTable: 'Load' metric contains a join of the '_LoadInternal' and
'DA_MaxAvServ' metrics.
```

The last metric is returned as the result for the 'Load' metric.

# GetTable

This element is used within an ExecutionDescriptor element and is equivalent to the following SQL operation:

Select * from T.

T is a metric.

```
<!ELEMENT GetTable EMPTY>
<!ATTLIST GetTable
     NAME CDATA #REQUIRED
     ASSOC_TARGET CDATA #IMPLIED
     METRIC_NAME CDATA #IMPLIED
     USE_CACHE (TRUE | FALSE | TRUE_IF_COLLECT) "FALSE">
```

## Attributes

NAME: Name of the metric.

ASSOC_TARGET: Target from which data is collected. This attribute is optional and when omitted, the METRIC_NAME points at a metric in the same target.

METRIC_NAME: Name of the metric that originates the request. If omitted, attribute NAME is used as METRIC_NAME.

USE_CACHE: Specifies whether the data can be fetched from the cache.

TRUE | FALSE (default) | TRUE_IF_COLLECT

## Elements

None

## Used In

ExecutionDescriptor

## Examples

```
<GetTable NAME="DiskActivity"/>
```

This statement gets all the columns of the 'DiskActivity' metric.

Refer to the example in ExecutionDescriptor.

# GetView

`GetView` creates a sub-table from a table. The newly created table is identified by the NAME attribute. It must be unique in the `ExecutionDescriptor`. This element is equivalent to the following SQL statement:

Select column1, column2,. . . from T

T is a metric.

```
<!ELEMENT GetView ((ComputeColumn | Column)*, (Filter | In)*)>
<!ATTLIST GetView
     NAME CDATA #REQUIRED
     FROM_TABLE CDATA #REQUIRED>
```

## Attributes

NAME: Name of the view.

FROM_TABLE: Table from which to generate the view.

## Elements

ComputeColumn

Column

Filter

In

## Used In

ExecutionDescriptor

## Examples

`GetView` may contain 0 or more instances of either 'ComputeColumn' or 'Column' elements and 0 or more instances of either 'Filter' or 'In' elements.

```
<GetView NAME="AvgSrvcTimeView" FROM_TABLE="DiskActivity">
                        <Column NAME="DiskActivityavserv"/>
        </GetView>
```

This is equivalent to the following SQL statement:

```
create view AvgSrvcTimeView as
select DiskActivityavserv from DiskActivity.
```

If no Column elements are present in `GetView,` all columns in the table are included.

Refer to the example in ExecutionDescriptor.

# Filter

Specifies the filter criteria. Filter is used to determine whether a row will be included in the new table. If a row does not satisfy any Filter criteria, it will be excluded.

```
<!ELEMENT Filter (#PCDATA)>
<!ATTLIST Filter
      COLUMN_NAME CDATA #REQUIRED
      SCOPE (GLOBAL | INSTANCE | SYSTEMGLOBAL) "GLOBAL"
OPERATOR (EQ | LT | GT | LE | GE | NE | CONTAINS | MATCH | ISNULL | ISNOTNULL)
"EQ">
```

## Attributes

COLUMN_NAME: Column name on which the filter criteria is to be applied.

SCOPE:

Supported values are:

- GLOBAL (default)
- INSTANCE
- SYSTEMGLOBAL
- OPERATOR: Specifies the operation to perform.

Supported operators are:

- EQ (default): Equal
- LT: Less than
- GT: Greater than
- LE: Less than or equal to
- GE: Greater than or equal to
- NE: Not equals
- CONTAINS: contains

- MATCH: matches
- ISNULL: is NULL
- SNOTNULL: is not NULL

## Elements

Character data representing the filter criteria.

## Used In

GetView

## Examples

```
<ExecutionDescriptor>
    <GetTable NAME="Servlet_raw" USE_CACHE="TRUE_IF_COLLECT"/>
    <GetView  NAME="result" FROM_TABLE="Servlet_raw">
        <Filter COLUMN_NAME="totalRequests1" OPERATOR="GT">0</Filter>
    </GetView>
</ExecutionDescriptor>
```

The result of the `ExecutionDescriptor` is the metric 'result' that has totalRequests1 > 0.

# Column

Represents a column to include in the new table.

```
<!ELEMENT Column EMPTY>
<!ATTLIST Column
     NAME CDATA #REQUIRED
     COLUMN_NAME CDATA #IMPLIED
     TABLE_NAME CDATA #IMPLIED>
```

## Attributes

NAME: Specifies the name of the column of a metric.

COLUMN_NAME: Specifies the name of the column. It can be omitted if it is same as NAME.

TABLE_NAME: Specifies the name of the metric. If Column is a part of the `GetView` element, this attribute must be excluded.

## Elements

None

## Used In

GetView

JoinTables

## Examples

```
<Column NAME="DiskActivityavserv"/>
```

'DiskActivityavserv' is the metric column that is selected for the operation.

```
<Column NAME="responseTime" TABLE_NAME="groupbyapps"/>
```

Column, 'responseTime' from 'groupbyapps' metric is selected for the operation.

Refer to the example in ExecutionDescriptor

# ComputeColumn

This element describes how to compute the values of a column.

```
<!ELEMENT ComputeColumn EMPTY>
<!ATTLIST ComputeColumn
      NAME                CDATA #REQUIRED
      EXPR                CDATA #REQUIRED
      IS_VALUE            (TRUE | FALSE) "FALSE"
DEFAULT_WHEN_EMPTY (TRUE | FALSE) "FALSE"
DEFAULT_VALUE       CDATA #IMPLIED>
```

## Attributes

NAME: Name of the column.

EXPR: Expression that is evaluated to calculate the value. Refer to the Examples of ColumnDescriptor for details about the expression grammar and usage.

IS_VALUE: If set to TRUE, the EXPR points to the actual string value else EXPR is the expression used to calculate the column.

TRUE | FALSE (default)

DEFAULT_WHEN_EMPTY:

TRUE | FALSE (default)

DEFAULT_VALUE: Default value for the column. If not specified the default value will be set to "0".

## Elements

None

## Used In

GetView

GroupBy

## Examples

```
<GetView NAME="NEW_TABLE" FROM_TABLE="ORIG_TABLE">
<Column NAME="cn1" COLUMN_NAME="A" />
```

```
<Column NAME="cn2" COLUMN_NAME="B" />
<ComputeColumn NAME="cn3" EXPR="cn1+cn2" />
<ComputeColumn NAME="cn4" EXPR="cn1/cn2" DEFAULT_WHEN_EMPTY="TRUE"
DEFAULT_VALUE="1"/>
<ComputeColumn NAME="isMultiThreaded.value" IS_VALUE="TRUE" EXPR="true" />
<Filter COLUMN_NAME="C" OPERATOR="EQ">3</Filter>
                </GetView>
```

Refer to the Examples of ColumnDescriptor for details about the expression grammar
and additional examples of expressions that are acceptable in the EXPR attribute.

# In

This element is equivalent of the SQL Statement select * from from_table where
column_name in ( select in_column_name from in_table_name)

```
<!ELEMENT In EMPTY>
<!ATTLIST In
     COLUMN_NAME CDATA #REQUIRED
     IN_TABLE_NAME CDATA #REQUIRED
     IN_COLUMN_NAME CDATA #REQUIRED>
```

## Attributes

COLUMN_NAME: Column name to search for.

IN_TABLE_NAME: Table in which to search for.

IN_COLUMN_NAME: Column name of the table specified in IN_TABLE_NAME
attribute.

## Elements

None

## Used In

GetView

# GroupBy

GroupBy will perform aggregation operation on a table/view to create a new table.
This element is equivalent to the SQL statement: Select sum(column_name) from
table_name

```
<!ELEMENT GroupBy (By*, (AggregateColumn | ComputeColumn)*)>
<!ATTLIST GroupBy
     NAME CDATA #REQUIRED
     FROM_TABLE CDATA #REQUIRED>
```

## Attributes

NAME: Specifies the name of the grouping.

FROM_TABLE: Specifies the name of the metric.

## Elements

By

AggregateColumn

ComputeColumn

## Used In

GetView

## Examples

This element may include 0 or more elements of 'By' and 0 or more elements of either `AggregateColumn` or `ComputeColumn`.

```
<GroupBy NAME="DA_MaxAvServ" FROM_TABLE="AvgSrvcTimeView">
```

This statement results in DA_MaxAvServ containing the result of the groupby operation applied to the AvgSrvcTimeView metric.

Refer to the example in ExecutionDescriptor.

# By

'By' element defines a column that constitutes a GroupBy clause. Each 'By' element will be added to the result table as a column.

```
<!ELEMENT By EMPTY>
<!ATTLIST By
      NAME CDATA #REQUIRED
      COLUMN_NAME CDATA #IMPLIED>
```

## Attributes

NAME: Name of the new column.

COLUMN_NAME: name for the new column.

## Elements

None

## Used In

GroupBy

## Examples

NAME must be unique within the new table. If COLUMN_NAME is not given, will be the same as NAME

# AggregateColumn

This element describes an operation to be performed on a column.

```
<!ELEMENT AggregateColumn EMPTY>
<!ATTLIST AggregateColumn
      NAME CDATA #REQUIRED
      COLUMN_NAME CDATA #REQUIRED
      OPERATOR (MAX | MIN | SUM | AVG | COUNT) "SUM">
```

## Attributes

NAME: Specifies the name of the AggregateColumn

COLUMN_NAME: Specifies the column name of the metric on which the operation is to be performed.

OPERATOR: Specifies the operation to be performed.

Supported operators are:

- MAX
- MIN
- SUM (default)
- AVG
- COUNT

## Elements

None

## Used In

GroupBy

## Examples

```
<AggregateColumn NAME="longestServ" COLUMN_NAME="DiskActivityavserv"
OPERATOR="MAX" />
```

This operation results in the calculation of the Maximum value from the 'DiskActivityavserv' column.

Refer to the example in ExecutionDescriptor.

# Union

Union element describes an operation to combine/merge two or more tables. Only tables with the same number of columns can be unioned together. This element is equivalent of the SQL statement:

```
Select * from T1 Union
Select * from T2
```

```
<!ELEMENT Union (Table+)>
<!ATTLIST Union
      NAME CDATA #REQUIRED
      DISTINCT (TRUE | FALSE) "FALSE">
```

## Attributes

NAME: Specifies the name for the union.

DISTINCT: If TRUE, any row that is exactly the same as previous row will be discarded.

TRUE | FALSE (default)

## Elements

Table

## Used In

ExecutionDescriptor

## Examples

Union element must include atleast 1 Table element.

```
<Union NAME="result4">
          <Table NAME="result" />
          <Table NAME="result1" />
          <Table NAME="result2" />
          <Table NAME="result3" />
      </Union>
```

'result4' will be a union of the 4 Tables. The resulting table will have the same column name list as first table.

> **✐ Note:**
>
> Only tables with the same number of columns can be unioned together. The newly created table is identified by NAME. It must be unique in the `ExecutionDescriptor`.

## Table

This element describes the metrics that can take part in Table operations like Union and JoinTables.

```
<!ELEMENT Table EMPTY>
<!ATTLIST Table
      NAME CDATA #REQUIRED>
```

## Attributes

NAME: Specifies the name of the metric.

## Elements

None

## Used In

Union

JoinTables

## Examples

```
<Table NAME="result" />
```

This example describes 'result' as a metric that can participate in Table operations.

Refer to the example of Union.

# JoinTables

This element describes the join operation. It is equivalent of the SQL statement:

Select C1, C2 where . . .

```
<!ELEMENT JoinTables (Table, Table+, (Column | ComputeColumn)*, Where*)>
<!ATTLIST JoinTables
      NAME CDATA #REQUIRED
      OUTER (TRUE | FALSE) "FALSE"
      BOTH_SIDE (TRUE | FALSE) "FALSE">
```

## Attributes

NAME: Specifies the name of the join.

OUTER: If true, specifies an OUTER join.

TRUE | FALSE (default).

BOTH_SIDE: If true, specifies BOTH_SIDE. The results would contain the same number of rows as a UNION.

TRUE | FALSE (default)

## Elements

Table

Column

ComputeColumn

Where

## Used In

ExecutionDescriptor

## Examples

JoinTables must include atleast 2 'Table' elements, 0 or more elements of either 'Column' or 'ComputeColumn' and 0 ore more elements of 'Where' element.

```
<JoinTables NAME="Load">
      <Table NAME="_LoadInternal"/>
      <Table NAME="DA_MaxAvServ"/>
   </JoinTables>
```

This example defines 'Load' as a join of '_LoadInterval' and 'DA_MaxAvServ' metrics.

Refer to the example in ExecutionDescriptor.

## Where

```
<!ELEMENT Where EMPTY>
<!ATTLIST Where
     FROM_TABLE CDATA #REQUIRED
     FROM_KEY CDATA #REQUIRED
     OPERATOR (EQ | GT | GE | LE | LT | NE | CONTAINS | MATCH) "EQ"
     JOIN_TABLE CDATA #REQUIRED
     JOIN_KEY CDATA #REQUIRED>
```

## Attributes

FROM_TABLE:

FROM_KEY:

OPERATOR: Specifies the operator in the where clause.

Supported operators are:

- EQ (default): Equals
- GT: Greater than
- GE: Greater than or equal to
- LE: Less than or equal to
- LT: Less than
- NE: Not equals
- CONTAINS: Contains
- MATCH: matches

JOIN_TABLE:

JOIN_KEY:

## Elements

None

## Used In

JoinTables

## Examples

```
<JoinTables NAME="all" OUTER="TRUE">
        <Table NAME="groupbyapps" />
        <Table NAME="groupbyappsejb" />
        <Column NAME="Application" TABLE_NAME="groupbyapps"/>
        <Column NAME="activeRequests" TABLE_NAME="groupbyapps"/>
        <Column NAME="responseTime" TABLE_NAME="groupbyapps"/>
        <Column NAME="activeMethods" TABLE_NAME="groupbyappsejb"/>
        <Column NAME="avgMethodExecTime" TABLE_NAME="groupbyappsejb"/>
                        <Where  FROM_TABLE="groupbyapps"
JOIN_TABLE="groupbyappsejb"
                                FROM_KEY="Application"
JOIN_KEY="ApplicationName"/>
</JoinTables>
```

# PushDescription

The push descriptor identifies a recvlet that is responsible for supplying data and/or events for a metric, and specifies data to be passed to that recvlet for each target. The name used for the recvlet here should match the recvlet name used in recvlets.reg.

```
<!ELEMENT PushDescriptor (ValidIf*, Property*) >
<!ATTLIST PushDescriptor
      RECVLET_ID CDATA #REQUIRED>
```

## Attributes

RECVLET_ID: Specifies the ID of the recvlet to use that is known to the EMAgent. This attribute must point to an element from the $ORACLE_HOME/lib/recvlets.reg file.

## Elements

ValidIf

Property

## Used In

Metric

## Examples

PushDescriptor may include 0 or more elements of 'ValidIf' and 'Property'.

```
<Metric . . .>
. . .
```

```
<PushDescriptor RECVLET_ID="AQMetrics">
<Property NAME="QueueName" SCOPE="GLOBAL">ALERT_QUE</Property>
<Property NAME="MachineName" SCOPE="INSTANCE">MachineName</Property>
<Property NAME="Port" SCOPE="INSTANCE">Port</Property>
<Property NAME="SID" SCOPE="INSTANCE">SID</Property>
<Property NAME="UserName" SCOPE="INSTANCE">UserName</Property>
<Property NAME="password" SCOPE="INSTANCE">password</Property>
<Property NAME="Role" SCOPE="INSTANCE">Role</Property>
<Property NAME="InstanceName" SCOPE="INSTANCE">InstanceName</Property>
<Property NAME="KeyField" SCOPE="GLOBAL">OBJECT_NAME</Property>
<Property NAME="KeyColumn" SCOPE="GLOBAL">name</Property>
</PushDescriptor>
                </Metric>
```

The properties included in the PushDescriptor are passed to the 'AQMetrics' receivelet. This receivelet provides data and/or events for the metric.

# Target Collection DTD Elements

Target Collection drives the background collection of metrics for the purposes of uploading their values in a central repository and/or the check of their values against specified conditions.

Note that the XML files conforming to this DTD will be generated by the system (Could be generated from Servlet Frontend or collector).

EMAgent can have more than one collection files each containing metrics that need to be collected for a particular target.

## TargetCollection

A TargetCollection describes the metric that needs to be collected for a particular target. This element applies to both default collections (type level) and instance specific collections. NAME, LEVEL and INCLUDE_DEFAULT attributes apply only to instance specific collections.

It can have 0 or more CollectionItem and CollectionLevel elements. The CollectionLevel element applies only to the default collections.

```
<!ELEMENT TargetCollection  ( CollectionLevel*, CollectionItem* ) >
<!ATTLIST TargetCollection
  TYPE CDATA #REQUIRED
  NAME CDATA #IMPLIED
  LEVEL CDATA #IMPLIED
  INCLUDE_DEFAULT (TRUE|FALSE)  "TRUE">
```

## Attributes

TYPE: Specifies the target type.

NAME: Specifies the name of the target. If this is the top-level element, NAME must not be null. If this file is included, it could be null. This attribute applies only to instance specific collections.

LEVEL: Specifies the collection level. The default will be the minimum. This attribute applies only to instance specific collections.

INCLUDE_DEFAULT: If set to TRUE, will include default collection with the same target TYPE. This attribute applies only to instance specific collections.

TRUE (default) | FALSE

## Elements

CollectionLevel

CollectionItem

## Used In

TargetCollection is a top-level element.

## Examples

```
<TargetCollection TYPE="preferred" >

<CollectionItem NAME = "Load" UPLOAD_ON_FETCH="TRUE">

<Schedule OFFSET_TYPE="INCREMENTAL">
<IntervalSchedule INTERVAL = "15" TIME_UNIT = "Min"/>
</Schedule>

<MetricColl NAME = "NfsOperations">
<Filter COLUMN_NAME="NfsCallsRate" OPERATOR="LE">100000</Filter>
<Condition COLUMN_NAME="NfsServPercentBadCalls" CRITICAL="10" WARNING="5"
OPERATOR="GT" OCCURRENCES="3" MESSAGE="NFS Bad Calls are %value%%."
MESSAGE_NLSID="netapp_filer_nfs_operations_nfs_per_bad_calls"/>
</MetricColl>

<MetricColl NAME = "CifsOperations">
<Filter COLUMN_NAME="CifsCallsRate" OPERATOR="LE">100000</Filter>
<Condition COLUMN_NAME="CifsPercentBadCalls" CRITICAL="10" WARNING="5"
OPERATOR="GT" OCCURRENCES="3" MESSAGE="CIFS Bad Calls %value%%."
MESSAGE_NLSID="netapp_filer_cifs_operations_cifs_per_bad_calls"/>
</MetricColl>

<MetricColl NAME = "SystemLoad" />

</CollectionItem>
</TargetCollection>

This example illustrates the preferred method of declaring CollectionItems.  One
collection item specifies multiple metrics with their own filter criteria and
Condition elements defined.

<TargetCollection TYPE="network" >

  <CollectionItem NAME = "Response">

    <Schedule>

      <IntervalSchedule INTERVAL = "300" TIME_UNIT = "Sec"/>

    </Schedule>

    <Condition COLUMN_NAME="Status"
```

```
                CRITICAL="ok"
                OPERATOR="NE"
                OCCURRENCES="3"
                MESSAGE="%target% adaptor is inaccesible or is connected."
                MESSAGE_NLSID="network_response_status"/>
    </CollectionItem>

</TargetCollection>
```

This example describes a default collection for 'network' target type. The metric is provided as the NAME attribute for the CollectionItem. This method of providing metric NAME in the CollectionItem is for backward compatibility only. Use the example with MetricColls instead.

TargetCollection includes optional 'CollectionLevel' element(s) and 'CollectionItem' element(s)

```
<TargetCollection TYPE="oracle_beacon" >
```

NAME attribute implies that this might not be in the default_collections.

# CollectionLevel

This element represents Collection Level List. It applies only to the default collections. The order implies the 'contains' relationship.

```
<!ELEMENT CollectionLevel EMPTY>
<!ATTLIST CollectionLevel
      NAME CDATA #REQUIRED>
```

## Attributes

NAME: the name of the collection level

## Elements

None

## Used In

[TargetCollection](#)

## Examples

```
<TargetCollection TYPE="example2"> <CollectionLevel NAME="LEVEL1"/>
<CollectionLevel NAME="LEVEL2"/> <CollectionLevel NAME="LEVEL3"/> . . .
<MetricColl NAME="metric1"> <ItemProperty NAME="prop1">foo</ItemProperty>
<Condition COLUMN_NAME="value" CRITICAL="bar" OPERATOR="EQ"/> </
MetricColl></TargetCollection>
```

The order for the collection level implies that LEVEL2 contains LEVEL1 collection items and LEVEL3 would include both LEVEL2 and LEVEL1 items.

Once the levels are declared using this element, the LEVEL attributes would refer to these levels.

# CollectionItem

A CollectionItem defines the collection of one or more metrics. It has a schedule.

```
<!ELEMENT CollectionItem (ValidIf*, Schedule?, (MetricColl+ | (ItemProperty*,
Filter*, LimitRows?, Condition*)  ) )>
<!ATTLIST CollectionItem
  NAME CDATA #REQUIRED
  LEVEL CDATA #IMPLIED
  UPLOAD CDATA "YES"
  UPLOAD_ON_FETCH (TRUE | FALSE) "FALSE"
  ATOMIC_UPLOAD (TRUE | FALSE) "FALSE"
  POSTLOAD_PROC CDATA #IMPLIED
  PRELOAD_PROC CDATA #IMPLIED
  CONFIG        (TRUE | FALSE) "FALSE"
  CONFIG_METADATA_VERSION CDATA #IMPLIED
  TIMEOUT  CDATA  #IMPLIED
  COLLECT_WHEN_DOWN (TRUE | FALSE) "FALSE"
  COLLECT_WHEN_ALTSKIP (TRUE | FALSE) "FALSE"
  COMBINE_WITH_OTHER_COLLECTION (TRUE | FALSE) "TRUE"
  PROXY_TARGET_NAME CDATA #IMPLIED
  PROXY_TARGET_TYPE CDATA #IMPLIED
  PROXY_TARGET_TZ   CDATA #IMPLIED
  PROXY_TARGET_TZ_REGION   CDATA #IMPLIED
  INITIAL_UPLOADS CDATA #IMPLIED
  DISABLED (TRUE | FALSE) "FALSE"
 REQUIRED (TRUE | FALSE) "FALSE">
```

## Attributes

NAME: Specifies the name of the collection.

LEVEL: The collection level.

UPLOAD: This attribute, if not specified, will be deemed as YES. NUMBER indicates how often the CollectionItem is uploaded. (Upload every 'n' collections).

YES (default) | NO | NUMERIC

UPLOAD_ON_FETCH: Collection Items marked as UPLOAD_ON_FETCH will have the same behavior as ATOMIC_UPLOAD with 1 difference – the upload occurs immediately.

TRUE | FALSE (default)

COLLECT_WHEN_DOWN: The default behavior is that the collection stops if the Response metric (if present for the target) indicates that the status of the target is down. The exception being the Response metric itself. But the behavior of not collecting when target is down can be overridden by setting this attribute to TRUE.

TRUE | FALSE (default)

COLLECT_WHEN_ALTSKIP: The default behavior is that the collection of metrics stops if an AltSkipCondition has been defined and there is a severity on the condition. Setting this attribute to TRUE allows collections to proceed even when this is the case. Note that a severity on the Response/Status condition is only overcome by using the COLLECT_WHEN_DOWN attribute.

TRUE | FALSE (default)

PROXY_TARGET_NAME: Used for proxy collection support, specifies Name of target data and severities should be uploaded for

PROXY_TARGET_TYPE: Used for proxy collection support, specifies Type.

PROXY_TARGET_TZ_REGION: Used for proxy collection support, specifies Timezone Region String (Eg "US/Pacific")

PROXY_TARGET_TZ: Used for proxy collection support, specifies Timezone (minutes from GMT: eg -420)

TIMEOUT: This is the time by which the metric evaluation is expected to finish. The time is provided in seconds. If the evaluation takes more than this time, the agent aborts the metric evaluation and returns a TIMEOUT exception. If this attribute is not provided or a value of zero, the agent defaults to twice the frequency of this metric evaluation in the collection file. Users can provide a time of less than zero to avoid any sort of timeout. A value less than 0 will force the agent to wait until the metric is evaluated completely.

POSTLOAD_PROC: Only applicable in UPLOAD_ON_FETCH situations. This attribute specifies an optional pl/sql procedures that the receiver should invoke when it receives the file with the contents of this collection.

PRELOAD_PROC: Only applicable in UPLOAD_ON_FETCH situations. This attribute specifies an optional pl/sql procedures that the receiver should invoke when it receives the file with the contents of this collection.

CONFIG: This attribute is used to tag collections of CONFIG metrics - these are handled specially by the Enterprise Manager framework. Note: Collection Items for CONFIG Metrics cannot specify ATOMIC_UPLOAD as FALSE. TRUE | FALSE (default)

INITIAL_UPLOADS: Defaults to 2, but can be set to a different number if more initial uploads need to be sent up before skipping uploads based on the UPLOAD parameter.

ATOMIC_UPLOAD: Collection Items marked as ATOMIC_UPLOAD will be bundled into a single file which will be uploaded in the regular upload interval (5 minutes).

TRUE | FALSE (default)

CONFIG_METADATA_VERSION: This attribute is used to specify version of CONFIG metrics.

COMBINE_WITH_OTHER_COLLECTION: Agent typically combines collections and executes them in a single thread sequentially to save on threads based on the interval. This can cause some delay in the metric execution if a previous one is taking some time. However some metrics would require to be executed on time. Setting this flag to FALSE would ensure that this metric is executed in its own thread.

TRUE (default) | FALSE

DISABLED: If set to TRUE, the agent will ignore this collection item.

REQUIRED: If set to TRUE, the console will disallow users from disabling this item.

## Elements

ValidIf

MetricColl

ItemProperty

Filter

LimitRows

Condition

## Used In

TargetCollection

## Examples

CollectionItem may contain 'ValidIf' element(s) which must all be satisfied for the CollectionItem to be evaluated. It may contain an optional 'Schedule' element and either a 'MetricColl' element or either 'ItemProperty', 'Filter', 'LimitRows' or 'Condition' elements.

For backward compatibility, a single metric can be specified using the NAME attribute, and its properties, filters and conditions can be provided as child elements.

The NEW preferred DTD has one or more Metric elements within a CollectionItem each indicating a metric to collect, and the filters, thresholds, etc. to associate with it.

```
<CollectionItem NAME="ProgramResourceUtilization">
```

This is an example of a simplest form of this element. 'ProgramResourceUtilization is a CollectionItem.

```
<CollectionItem NAME = "general_collection" UPLOAD="12"
INITIAL_UPLOADS="4">
```

In this sample, 'general_collection' will be uploaded 4 times (number dictated by INITIAL_UPLOADS) initially and after that it will be uploaded once every 12 times (number dictated by UPLOAD attribute).

```
<CollectionItem NAME = "Inventory" UPLOAD_ON_FETCH = "TRUE" TIMEOUT =
"3600">
```

'Inventory' collection item will be uploaded when the metrics are collected. Timeout specified is 1 hour.

```
<CollectionItem NAME = "oracle_security" UPLOAD_ON_FETCH = "TRUE" CONFIG =
"TRUE">
```

'oracle_security' in this sample, involves collecting CONFIG metrics and should be indicated as such to the EMAgent.

```
<CollectionItem NAME = "UserResourceUsage" UPLOAD="YES"
UPLOAD_ON_FETCH="FALSE" COLLECT_WHEN_DOWN="FALSE">
```

The UPLOAD attribute can take a 'Yes', 'No' or a numeric value. 'UserResourceUsage' collection will be attempted even when the response metric indicates that the target is down.

The PROXY_TARGET_TZ_REGION takes precedence over PROXY_TARGET_TZ if both are specified

Refer to the example in TargetCollection.

# MetricColl

The MetricColl element refers to a metric that is being collected within a collection item.

```
<!ELEMENT MetricColl (ItemProperty*, Filter*, LimitRows?, Condition*)>
<!ATTLIST MetricColl
  NAME CDATA #REQUIRED
  TRANSIENT (TRUE|FALSE) "FALSE"
  UPLOAD_IF_SEVERITY (WARNING|CRITICAL|CHANGE_ONLY) "CHANGE_ONLY">
```

## Attributes

NAME: This is the name of the metric to collect.

TRANSIENT: If this attribute is set to TRUE would indicate that the data of this metric should be uploaded or is collected to refresh the cache used in the evaluation of other metrics.UPLOAD_IF_SEVERITY: Only effective when UPLOAD=NO and UPLOAD=N>1.

Supported values are:

- CHANGE_ONLY (default): Upload data when there is severity change
- WARNING: Upload data when there is severity change and when any condition is in WARNING or CRITICAL
- CRITICAL: Upload data when there is severity change and when any condition is in CRITICAL

## Elements

ItemProperty

Filter

LimitRows

Condition

## Used In

CollectionItem

## Examples

MetricColl may include optional ItemProperty element(s), Filter element(s), a single optional LimitRows element and optional Condition element(s).

```
<MetricColl NAME = "WebServicesService"/>
```

This is the typical usage for MetricColl. 'WebServicesService' metric is associated with a CollectionItem.

Refer to the example in TargetCollection.

# LimitRows

LimitRows is a filtering mechanism that can be applied to the collected data, before the data is sent to the repository via the Upload Manager. It limits the number of rows hat are uploaded.

```
<!ELEMENT LimitRows EMPTY>
<!ATTLIST LimitRows
          COLUMN_NAME CDATA #IMPLIED
          SORT_ORDER (ASCEND|DESCEND|NO_ORDER) "NO_ORDER"
          LIMIT_TO    CDATA   #REQUIRED>
```

## Attributes

COLUMN_NAME:

SORT_ORDER:

Supported values are:

- ASCEND:
- DESCEND:
- NO ORDER (default):

LIMIT_TO: Specifies the limit for the number of rows in the collection.

## Elements

None

## Used In

CollectionItem

MetricColl

## Examples

```
<LimitRows LIMIT_TO="16" />
```

This example limits the collection results to 16 rows.

# ItemProperty

This element describes a name value pair for a property.

```
<!ELEMENT ItemProperty (#PCDATA)>
<!ATTLIST ItemProperty
  NAME CDATA #REQUIRED
ENCRYPTED (NA|FALSE|TRUE) "NA"
>
```

## Attributes

NAME: Name of the Item property

ENCRYPTED: Indicates whether the property value will be encrypted.

The following values are defined for the attribute:

NA (default): Encryption is not available. The agent will not attempt to encrypt the value.

FALSE: Encryption is available. The agent will attempt to encrypt the value.

TRUE: Encryption is available. The agent has encrypted the value.

## Elements

Character data

## Used In

CollectionItem

MetricColl

## Examples

```
<TargetCollection TYPE="example2">
   . . .

   <MetricColl NAME="metric1">
      <ItemProperty NAME="prop1">foo</ItemProperty>
      <Condition COLUMN_NAME="value" CRITICAL="bar" OPERATOR="EQ"/>
   </MetricColl>
</TargetCollection>
```

'prop1', ItemProperty will be utilized to compute the value of 'prop1' property defined in 'USER' scope in 'metric1' in TargetMetadata for target type 'example2'.

# Filter (for Target Collection)

Filter defines a filtering mechanism that can be applied to the collected data before the data is sent to the repository via the Upload manager. If filtering is not applied, all the data that is collected through a Fetchlet is sent to the repository. As a result the repository can get filled quickly when uploading certain metrics. To alleviate this problem, filtering mechanism is applied to the data before uploading. The filter criteria are specified in collection xml.

```
<!ELEMENT Filter (#PCDATA)>
<!ATTLIST Filter
     COLUMN_NAME CDATA #REQUIRED
     OPERATOR  (EQ|LT|GT|LE|GE|NE|CONTAINS|MATCH) "EQ"
     AFTER_SEVERITY_CHECKING (TRUE|FALSE)  "FALSE" >
```

> **✎ Note:**
>
> Filter elements for TargetCollection and TargetMetadata are different.

## Attributes

COLUMN_NAME: Name of the column in the metric to filter on.

OPERATOR: Specifies the operation

EQ (default): Equals

LT: Less than

GT: Greater than

LE: Less than or equal to

GE: Greater than or equal to

NE: Not equals

CONTAINS: Contains

MATCH: Matches

AFTER_SECURITY_CHECKING: If TRUE, filter will be applied after severity checking.

TRUE | FALSE (default)

## Elements

Character data

## Used In

CollectionItem

MetricColl

## Examples

```
<Filter COLUMN_NAME="total_connections" OPERATOR="NE">0</Filter>
```

The result will include only those rows where 'total_connections' not equal to 0.

## Condition

Condition element defines when a severity will be triggered.

```
<!ELEMENT Condition (CategoryValue*, KeyColumn*, FixitJob? )>
<!ATTLIST Condition
  CRITICAL CDATA #IMPLIED
  WARNING CDATA #IMPLIED
  OPERATOR (EQ | LT | GT | LE | GE | NE | CONTAINS | MATCH ) "GT"
  OCCURRENCES CDATA "1"
```

```
    NO_CLEAR_ON_NULL (TRUE │ FALSE) "FALSE"
    MESSAGE CDATA #IMPLIED
    MESSAGE_NLSID CDATA #IMPLIED
    COLUMN_NAME CDATA #REQUIRED
    PUSH (TRUE │ FALSE) "FALSE"
    GENERATE_INIT_CLEAR (TRUE │ FALSE) "FALSE"
    ALERT_CONTEXT CDATA #IMPLIED
    CLEAR_MESSAGE CDATA #IMPLIED
    CLEAR_MESSAGE_NLSID CDATA #IMPLIED
    STATELESS_ALERT (TRUE │ FALSE)  "FALSE"
>
```

If it is Table Metric, MetricColumn is used to define which column and key to use to identify the row and column.

If KEYONLY_THRESHOLDS is set to TRUE for a Metric column, the Condition element must include a KeyColumn element.

## Attributes

CRITICAL: Threshold. A special value, "NotDefined" for the threshold ensures that the result of the operation specified by the OPERATOR will fail.

WARNING: Threshold. "NotDefined" may also be applied to WARNING.

OPERATOR: Specifies the operation to evaluate the condition.

EQ: Equals

LT: Less than

GT (default): Greater than

LE: Less than or equal to

GE: Greater than or equal to

NE: Not equals

CONTAINS: Contains

MATCH: Matches

OCCURRENCES: The default value is 1.

NO_CLEAR_ON_NULL: This attribute is used to control severity clearing when a null value is returned for a metric column. It defaults to FALSE with the behavior that a null value ends up clearing previous alert severities. With a TRUE value for this attribute, null values will be skipped in severity evaluations without clearing the severity.

MESSAGE: The message attribute is a message template that will be used to generate messages to be sent with the event occurrence. This message can contain the following place holders.

- %value%: The value of the metric (or column of metric)
- %target%: name of the target
- %metric_id%: metric id
- %column_name% This will be the value of any column this can include value columns as well as key columns

- %warning_threshold%: the warning threshold of the condition

- %critical_threshold%: the critical threshold of the condition

- %num_of_occur%: number of occurrences

MESSAGE_NLSID: Specifies the String ID of the ResourceBundle for the message.

*COLUMN_NAME: For table metric, COLUMN_NAME defines which column will be checked. KeyColumn will be used to identify a row.

PUSH: This attribute is used to distinguish conditions created for push-based alerts from conditions that are evaluated over the collected data. The agent does not evaluate PUSH="TRUE" conditions.

GENERATE_INIT_CLEAR: This attribute can be used to override the agent's behavior of not generating a severity the very first time a CLEAR is generated. Set this to TRUE if you do want the initial CLEAR.

ALERT_CONTEXT: This attribute will be used to pass the related alert context. This new attribute may contain a list of column names separated by ";".

CLEAR_MESSAGE: Specifies a different message when an alert is cleared. If this attribute is missing then the MESSAGE attribute is used when alerts are cleared.

CLEAR_MESSAGE_NLSID: Specifies the NLSID for the clear message. If absent, the MESSAGE_NLSID is used when alerts are cleared.

STATELESS_ALERT: The default is false. If this attribute is set to TRUE, it indicates to Enterprise Manager not to retain any state associated with the condition. The default is False.

## Elements

CategoryValue

KeyColumn

FixitJob

## Used In

CollectionItem

MetricColl

## Examples

Condition element may include optional CategoryValue element(s), KeyColumn element(s) and an optional FixitJob element.

If the result after the keys are applied contains more than one row, the event occurrence generated will have content/message for each row that has crossed the threshold.

MATCH is used for regular expression.

For example:

```
OPERATIOR="MATCH" CRITICAL=".*ORA.*ERR.*"
```

This statement will match a string containing both ORA and ERR such as "ORA-ERR 345".

CategoryValue sub tags are used to classify the Condition along two axis, CLASS and CATEGORY. For e.g. CLASS=Fruits, CATEGORY=RedFruits Categorization of Conditions is useful for Root Cause Analysis among other things.

```
<Condition COLUMN_NAME="alertSeverity"
            CRITICAL="NotDefined"
WARNING="NotDefined"
OPERATOR="LE"
            OCCURRENCES="1"
            MESSAGE="%alertConcatString%"
NO_CLEAR_ON_NULL="TRUE"
            MESSAGE_NLSID="host_alertLog_alertSeverity_cond"/>


<Condition COLUMN_NAME="State"
CRITICAL="open"
            OPERATOR="EQ"
            MESSAGE="%Description%"
            ALERT_CONTEXT="In-contextLaunchURL"
            NO_CLEAR_ON_NULL="TRUE" />
```

These are examples of the Condition element.

```
<TargetCollection TYPE="examplec1" >
<CollectionItem NAME = "Response">
    <Schedule>
      <IntervalSchedule INTERVAL = "300" TIME_UNIT = "Sec"/>
    </Schedule>
    <Condition COLUMN_NAME="Status"
              CRITICAL="ok"
              OPERATOR="NE"
              OCCURRENCES="3"
              MESSAGE="%target% adaptor is inaccesible or is connected."
              MESSAGE_NLSID="network_response_status"/>
</CollectionItem>

</TargetCollection>
```

This sample gives the context for the Condition element.

For additional examples refer to TargetCollection example.

# KeyColumn

The KeyColumn element is used to specify the key column for a table. It identifies a row of a table. This element must be present in a Condition element if KEYONLY_THRESHOLDS attribute is set for a Metric column.

```
<!ELEMENT KeyColumn (#PCDATA)>
<!ATTLIST KeyColumn
  COLUMN_NAME CDATA #REQUIRED
  OPERATOR    (EQ | LIKE) "EQ">
```

## Attributes

COLUMN_NAME: Specifies name of the key column

OPERATOR:

Supported values are:

- EQ (default) : Equals
- LIKE: like

## Elements

Character data

## Used In

[Condition](#)

## Examples

```
<Condition COLUMN_NAME="col3" WARNING="def" OPERATOR="CONTAINS">
      <KeyColumn COLUMN_NAME="col1">keyA</KeyColumn>
</Condition>
```

# FixitJob

This element describes the action to be taken in response to an alert.

```
<!ELEMENT  FixitJob  (Property*) >
<!ATTLIST  FixitJob
      TYPE    CDATA  #IMPLIED >
```

## Attributes

TYPE: Specifies the type of the job.

## Elements

[Property](#)

## Used In

[Condition](#)

## Examples

```
<TargetCollection TYPE="examplec1" >

  <CollectionItem NAME = "FixitExample" UPLOAD_ON_FETCH="TRUE">
    <Schedule>
      <IntervalSchedule INTERVAL="60" TIME_UNIT="Sec" />
    </Schedule>

     <MetricColl NAME="metric1"/>

<Condition COLUMN_NAME="col2" CRITICAL="0" OPERATOR="GT" OCCURRENCES="1">
       <FixitJob TYPE="OSCommand">

            <Property NAME="prop_env" SCOPE="ENV">EMDROOT</Property>
            <Property NAME="prop_loc" SCOPE="INSTANCE">FILE_LOC</Property>
```

```
            <Property NAME="COMMAND" SCOPE="GLOBAL">rm %prop_loc %</Property>
        </FixitJob>
      </Condition>
  </CollectionItem>
</TargetCollection>
```

This example illustrates a simple FixitJob that deletes files in response to a condition. The COMMAND property specifies the command that is executed when the value in col2 of the metric triggers the condition.

# A

# Out-of-Box Associations

Enterprise Manager provides a common set of association types that should meet the needs of most plug-in developers. As a plug-in developer, you are encouraged to become familiar with these association types and use them if applicable. As a plug-in developer, you should also update the Table of Integrators and Documents with links to the documents describing your association types and your usage of all association types (allowed_pairs).

The following tables provide details on the out-of-box associations:

- Table A-1
- Table A-2
- Table A-3
- Table A-4
- Table A-5
- Table A-6
- Table A-7
- Table A-8
- Table A-9
- Table A-10
- Table A-11
- Table A-12
- Table A-13
- Table A-14
- Table A-15
- Table A-16
- Table A-17
- Table A-18
- Table A-19
- Table A-20

**Table A-1    application_contains**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: Contains<br><br>Core/Extended: Core<br><br>Display Name: "contains (app component)"<br><br>Inverse:member _of_application | Source: any aggragate type which is not kind of composite, for example oracle_emrep. For the source type of composite type, user should use composite_contains or assoc types extended from it.<br><br>Destination: member entity types, for example, j2ee_applications for Enterprise Manager console and backend service are members for oracle_emrep<br><br>Cardinality: 0..* | Capture the membership between application and its members. The source is an aggregation of the members. One member can be part of multiple aggregations. | Indicate a membership of an application, Can be used in the topology. |

**Table A-2    app_composite_contains**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: application_cont ains, composite_conta ins<br><br>Core/Extended: Core<br><br>Display Name: "contains (app_composite component)"<br><br>Inverse:member _of_composite_a pp | Source: any applicaion that is also a composite type, for example siebel_server. For the source type of cluster type, user should use cluster_contains<br><br>Destination: member entity types, for example, siebel_component_gro up is a member of siebel_server<br><br>Cardinality: 0..* | Capture the membership between application and its members. The membership is also a kind of composition. One member can be part of only one composition | Indicate a membership of an application, Can be used in the topology. |

**Table A-3    authenticated_by**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: depends_on<br><br>Core/Extended: Core<br><br>Display Name: "authenticated_by)"<br><br>Inverse:authenticates | Source: An ME that requires authentication<br><br>Destination: Me providing the Authentication, for example: oracle_ldap<br><br>Cardinality: 0..1 | Capture the membership between application and its members. The membership is also a kind of composition. One member can be part of only one composition | Indicate a membership of an application, Can be used in the topology. |

**Table A-4    composite_contains (abstract)**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: contains,uses<br><br>Core/Extended: Core<br><br>Display Name: "CompositeContains"<br><br>Inverse: member_of_composite | Source: Source: any composite types<br><br>Destination: Members of the composite.<br><br>Cardinality: 0..* | A form of aggregation which requires that a part instance be included in at most one composite at a time. For example: If a database D1 is part of Oracle RAC cluster R1, it cannot be part of another cluster R2. This is used to place a box around the source and all its members to indicate that the members cannot be part of another source.<br><br>The restriction applies to specific concrete association type extended from composite_contains. An ME can be a destination of no more than 1 assoc of type T if T extends composite-contains. But an ME can be destination with different source ME as long as the concrete composit_contains types are different. | Current usage is to link cluster targets (rac_database,cluster, weblogic_cluster) to its members. Framework functionality, such as topology viewer, can use it. |

**Table A-5    cluster_contains**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: composite_contains<br><br>Core/Extended: Core<br><br>Display Name: "contains (in cluster)"<br><br>Inverse:member_of_cluster | Source: A cluster target type, such as RAC or Cluster<br><br>Destination: Member type of the cluster. The cluster member types should be the same<br><br>Cardinality: 1..* | Cluster membership, where the members are of the same type and together provide scalability and redundancy. Also indicates composite containment; Cluster A cluster_contains B implies that B cannot be member of another cluster C. | Tools like Consolidation Planner need to know cluster membership. Also, all the general tools such as Topology Viewer. |

**Table A-6    connects_through**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: depends_ on<br><br>Core/Extended: Core<br><br>Display Name: "connects_through"<br><br>Inverse: connects | Source: An ME which is connecting to another ME via a intermediate path. Example: an application connecting to database via listener<br><br>Destination: Me providing the access point for another ME ( oracle_listener, oracle_apache, slb)<br><br>Cardinality: 0..* | Application connects_through Listener, which exposes database.Service connects_through oracle_apache, which exposes oracle_oc4j. | Used in Applications topology to represent the connection to the end via an intermediate path. Example: connects_through listener, which exposes database.The source will also have a direct functional dependency directly on the end point ( application stores_on_db database)In functional view, the direct dependency of stores_on_db will be shown, in physical view, the listener link will be shown. |

**Table A-7    contains (abstract)**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: none<br><br>Core/Extended: Core<br><br>Display Name: "contains"<br><br>Inverse: member_of | Any source ME and its member ME types<br><br>Cardinality: 0..* | Indicates containment membership. A contains B implies that B is one of the parts that make up A.<br><br>All containment relationships should be captured by plug-in developers except that system membership is captured by the OMS and TC containment is not required to be represented via an assoc instance. You must use the concreted types, which extend from "contains" | However, user can query instances for all the association types which extend from the "constains". Framework functionality, such as topology viewer, can query this type of associations. |

**Table A-8    depends_on(abstract)**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: uses<br><br>Core/Extended: Core<br><br>Display Name: "Depends on"<br><br>Inverse: depended_on_by | Source: any ME type<br><br>Destination: any ME type<br><br>Cardinality: 0..* | For any ME A that depends on ME B for its availability. If ME B is not available, the availability of A may be impacted. | Framework functionalities, such as RCA and Topology Viewer, can use it. |

**Table A-9    deployed_on**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: runs_on, member_of_application<br><br>Core/Extended: Core<br><br>Display Name: "Deployed on"<br><br>Inverse: deploys | Source: any ME except target component<br><br>Destination: j2ee container<br><br>Cardinality: 1..* | Application A is deployed into a J2EE? container B. | Topology viewer can display the application deployed on a j2ee server . |

**Table A-10    exposes**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: uses<br><br>Core/Extended: Core<br><br>Display Name: "exposes"<br><br>Inverse: exposed_by | Source: ME providing the access point for another ME ( oracle_listener, oracle_apache, slb)<br><br>Destination: ME of target being accessed via the source.<br><br>Cardinality: 0..* | Some ME's functionality is exposed through other ME, such as oracle_listener exposes oracle_database to application, oracle_apache expose oc4j.Capture more semantics for uses: What the listener can do gets impacted if the db goes down, but listener itself does not go down, it is degraded mode.<br><br>Used to represent the entry points for targets/ systems.<br><br>Listener exposes oracle_database. In this case it is strictly not the database target which is providing the accesspoint, the oracle database can be thought of providing the services which are available via the listener.<br><br>oracle_http exposes oc4j. Here again, the oc4j target may not be providing the http service, the http service could be running as a seperate service which lets an application connect to oc4j. | Framework functionality, such as topology viewer, can use it. |

**Table A-11    hosted_by**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: runs_on<br><br>Core/Extended: Core<br><br>Display Name: "hosted_by"<br><br>Inverse: host_for | Source: any ME type except a system, service, or target component<br><br>Destination: host<br><br>Cardinality: 1 | For any target T that is hosted_by H, the process(es) that comprise T execute on host H.A target can be hosted by only one host. | Used to locate the targets running on the given host. |

**Table A-12    installed_at**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: uses<br>Core/Extended: Core<br>Display Name: "installed_at"<br>Inverse: install_home_for | Source: any ME except target component<br>Destination: A ME representing an install home<br>Cardinality: 1 | A installed at B indicates that B represents the install home for A Example: oracle database --> installed_at --> oracle_home. | Used to denote the link to the install home where the software for the ME is installed.Used in patching to get to the oracle home where the target is installed. |

**Table A-13    internal_contains (for internal OMS use only)**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: contains<br>Core/Extended: Core<br>Display Name: "InternalContains"<br>Inverse: internal_member _of | Source: System/Group<br>Destination: A system can contain any ME except group, a group can contain any ME except target component<br>Cardinality: 0..* | A special form of an association that specifies a whole-part relationship between the system and a component part. The component part can exist independent of the system and can be part of multiple systems. System A contains B implies that B is one of the parts that make up A. B can also be included in other system C | User should normally use Group/System API to find the members. But use can query the 'internal_contains' against table/view. |

**Table A-14    managed_by**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: monitored_by, uses<br>Core/Extended: Core<br>Display Name: "managed_by"<br>Inverse: manages | Source: any ME type except target component<br>Destination: an ME type that can provide management functionality for other ME types. For example, oracle_cs can manage oracle_database, oracle_listener etc.<br>Cardinality: 1..* (A specific allowed_pair can have stricter cardinality, such as 1) | The destination ME may work as watchdog and can start source target. The entity which manages another entities can make change to the managed entities, which the monitored_by doesn't have this semantic. | Framework functionality, such as topology viewer, can use the association |

**Table A-15    monitored_by**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends:<br>Core/Extended: Core<br>Display Name: "monitored_by"<br>Inverse: monitors | Source: any ME type except target component<br>Destination: agent<br>Cardinality: 1 (cardinality is 1 at any given moment) | For any target T that is monitored by an agent. Example: target T--> monitored_by -> agent A . | Used in agent syncrononzation/ availability calculations/ framework code |

**Table A-16    provided_by**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: depends_on<br>Core/Extended: Core<br>Display Name: "provided_by"<br>Inverse: provides | Source: an ME, typically representing some kind of service, such as the db service, fusion product, webservice<br>Destination: A system or a target which is providing the service<br>Cardinality: 1 | A provided_by B indicates that the service of B is provided by ME A. | Topology viewer can display the association |

**Table A-17    runs_on (abstract)**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: depends_on<br>Core/Extended: Core<br>Display Name: "runs_on"<br>Inverse: runs | Source: any ME except target component<br>Destination: any ME which provides infrastructure for other MEs to run, such as VM.<br>Cardinality: 1 | A run_on B indicates that B provides infrastructure for some processes of A to execute. Note: processes is used in the English sense and does not indicate OS processes. | Framework functionality, such as topology viewer, can use it. |

**Table A-18    stores_on**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: depends_on<br>Core/Extended: Core<br>Display Name: "stores_on"<br>Inverse: stores | Source: typically a target which stores data. For example: oracle_database or sql server.<br>Destination: an ME representing storage. For example: netapp filer or exadata.<br>Cardinality: 0..* | Indicates the link to the target representing the storage of the bits. A stored_on B indicates that B provides infrastructure for storage of bits of A, Example: datafile-->stored_on-->netapp_filer. The stored data can be static or updatable. | Used to denote the link to the storage infrastructure. This is to visually locate the storage details in the topology. |

**Table A-19    stores_on_db**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: stores_on<br><br>Core/Extended: Core<br><br>Display Name: "Data Repository"<br><br>Inverse: data_repository_for | Source: an ME that stores data in a database.<br><br>Destination: ME providing the database repository for storing the data Example: application stores_on_db oracle_database.<br><br>Cardinality: 0..* | Represents depends_on in that if the database server is down, the source can be down. | Used in Applications topology to represent the database where an applications data is stored. |

**Table A-20    uses (abstract)**

| Basic Details | Source/Destination | Description | Usage |
|---|---|---|---|
| extends: None<br><br>Core/Extended: Core<br><br>Display Name: "uses"<br><br>Inverse: used_by | Source: any ME type<br><br>Destination: any ME type<br><br>Cardinality: 0..* | For any ME A that depends on ME B for its working but does not affect availability. | Used in topology pages to indicate non availability dependency |

# B

# Plug-in Technical Checklist

Every metadata plug-in is assessed for quality and best practices using the Self Validation checklists included in this appendix.

Oracle recommends that you self-validate your plug-in against these checklists before submitting your plug-in for a formal review. Alternatively contact the Enterprise Manager Release Management team for the latest version of this checklist.

It includes the following sections:

- Checking your Plug-in
- Checking Targets
- Checking Customized UIs
- Checking Job Types
- Checking Reports
- Testing your Plug-in

## Checking your Plug-in

Table B-1 provides a list to check the plug-in data.

For more information about defining plug-in metadata, see Defining the Plug-in .

**Table B-1    Plug-in Metadata Checklist**

| Category | Checklist Item |
|---|---|
| Readme Tag | Include a Readme tag in the plugin.xml file that provides a description of your plug-in. Ensure that it is at least 80 characters in length. <br><br> For more information, see Table 2-1 |
| Display Name Attribute | Include a DisplayName attribute in the plugin.xml file such as: <br><br> `<PluginAttributes DisplayName="Oracle Sun ZFS Storage Appliance" Type="MP"/>` <br><br> For more information, see Table 2-1 |
| TargetTypeList Tag | Include a TargetTypeList tag in the plugin.xml file such as: <br><br> `<TargetTypeList> <TargetType isIncluded="TRUE" name="sun_storage_7000" />` |

**Table B-1    (Cont.) Plug-in Metadata Checklist**

| Category | Checklist Item |
|---|---|
| Version Support Information | If you are supporting a specific version or specific range of versions for these targets, ensure you define these versions. If you do not specify versions, then Enterprise Manager assumes that this plug-in can monitor and manage all versions of the target type.<br><br>**Note**: It is not a mandatory requirement to specify supported versions, but check whether you need it.<br><br>The following example is from a database plug-in:<br><br>`<TargetType name="oracle_database">`<br>`<VersionSupport> <SupportedVersion supportLevel="Comprehensive"`<br>`minVersion="9.2.0.8.0" maxVersion="9.2.0.8.0"/>`<br>`<SupportedVersion supportLevel="Comprehensive" minVersion="10.1.0.5.0"`<br>`maxVersion="10.1.0.5.0"/>`<br>`<SupportedVersion supportLevel="Comprehensive" minVersion="10.2.0.4.0"`<br>`maxVersion="10.2.0.5.0"/>`<br>`<SupportedVersion supportLevel="Comprehensive" minVersion="11.1.0.7.0"`<br>`maxVersion="11.1.0.7.0"/>`<br>`<SupportedVersion supportLevel="Comprehensive" minVersion="11.2.0.1.0"`<br>`maxVersion="11.2.0.3.0"/>`<br>`</VersionSupport>`<br><br>For more information, see Table 2-1 |
| Specific Plug-in Category | Choose a specific category other than "Others". Contact the Oracle Extensibility reviewers if your plug-in does not fit into the following predefined categories.<br>• Applications<br>• Databases<br>• Middleware<br>• Engineered Systems<br>• Cloud<br>• Servers, Storage and Network<br>For more information, see Table 2-1 |
| Generic Plug-in | The default is Generic, so Oracle recommends removing the `<PluginOMSOSAruId>` tag or define as follows:<br><br>`<PluginOMSOSAruId value="2000"/>`<br><br>For more information, see Table 2-1 |
| Size of the OPAR less than 2 MB | Ensure that your Oracle Plug-in Archive (OPAR) file is less than 2 MB.<br>For more information about the OPAR file, see Creating the Plug-in Archive. |
| Plug-in Validation Report | Ensure that no violations are reported in the plug-in validation report.<br>For more information, see Validating the Plug-in. |
| Repository Connection Details | Ensure that the plug-in validation report does not show any skipped validations because of missing repository connection details. |

# Checking Targets

Table B-2 provides a checklist which applies to defined targets for your plug-in. This checklist is only applicable if you have files in the *plugin_stage*/oms/metadata/ targetTypes directory.

For more information about targets, see Creating Target Metadata Files and Collecting Target Configuration Data.

**Table B-2    Targets Checklist**

| Category | Checklist Item |
| --- | --- |
| Target Type Name | Ensure that the target type name follows the pattern *company_plugin_tag_type _name*, such as oracle_vt_zone or oracle_emas_forms_server |
| Target is an Entity | Ensure that the target is a monitorable and manageable entity and it makes business sense to model it. |
| Identifiable Presence | Ensure that the target being modelled has a identifiable presence even if Enterprise Manager is not installed. |
| Manageable Entity Class | Identify the manageable entity class to which the target type belongs and set the property accordingly:<br>• is_system<br>• is_end_user_system<br>**Note**: This property is for end-user systems. Most plug-ins do not require this so check with the Oracle Extensibility reviewers before setting the property.<br>• is_service<br>• is_install_home<br>• is_group<br>**Note**: This property should be set the target_type=composite group only.<br>• is_existence_only<br>**Note**: This property should be set for new targets that are not fully managed or monitored.<br>For more information, see Table 3-2. |
| Monitoring Mode | This check applies to repository-side targets only. Ensure that MonitoringMode is set to Repository for Repository target types. |
| Monitoring Mode | This check applies to MultiAgent-side targets only.<br>Set MonitoringMode set to OMSMediated for multi-agent target types. |
| Response Metric | All target types must have a response metric defined.<br>For more information, see Defining the Basic Response Metric Group. |
| Target Type Metadata Version | The Target Type metadata version consists of 2 numbers, *MM.NN* where *MM* is the major version number and *NN* is the minor version number.<br>Set the major version number to the main plug-in release, such as 12 if the plug-in release is 12.1.*N.N.N* and 13 if the plug-in version is 13.*N.N.N.N*.<br>Set the minor version number to 2 digits starting with 01, such as 13.01 to start with for 13.*x* plug-ins<br>**Note**: You do not have to update the metadata version if you are changing query or execution descriptor or the agent-side script.<br>For more information, see Defining the Target Type Metadata. |

**ORACLE**

**Table B-2    (Cont.) Targets Checklist**

| Category | Checklist Item |
| --- | --- |
| Target Type Metadata Minor Version | Ensure that the minor version number uses a 2 digit format, such as 13.01 instead of 13.1. |
| | **Note**: Minor versions are compared right-padded to 20 digits, so 13.9 > 13.10 in meta version semantics, since 13=13 and 9 right padded to 20 digits is greater than 10 right padded to 20 digits (900000... > 100000..). The number of digits in the minor version must be consistent when you move from one version to another. Having 2 digits in the minor version allows you to bump up until 99. |
| | DDR patches supplied on top of previous releases use the format 13.NN*YYWWW*. If you are providing a metadata patch on top of 13.01, then the patch version is 13.0113005 (for 5th week of 2013). 13.0113005 is greater than 13.01 but less than 13.02 in target type metadata version semantics |
| | For more information, see Defining the Target Type Metadata. |
| Associations | Ensure that no abstract association types are used (`select assoc_type from mgmt_assoc_types where is_abstract=1`). |
| Associations | Ensure that only core association types are used (`select assoc_type from mgmt_assoc_types where category=1`) |
| Associations | Ensure that the Provided_by/relies_on_key_component allowed pair is not defined between the service and any other target type. |
| Valid Target Properties | Ensure that the Target properties include only properties that are used for monitoring the target, such as collecting a metric. |
| | Do not use target properties as a name value pair to dump data against the target. If it is not actively used by the Management Agent, then it is not a target property. |
| Target Version Property | Ensure that the Target version property is added. |
| | The Target version property captures the target version. By default, Enterprise Manager uses the "version" property to represent the target version. |
| | Target version is required for managing the target so that you can indicate that a particular version of the target is deprecated with a new release of plug-in. It helps administrators to determine the versions of the products they are using also. |
| Credential Sets | Ensure that credentials are defined as Credential Sets. |
| | For more information, see Defining Credentials. |
| Storing Credentials | Do not store credentials (user name or password) in target properties. They must be modelled as credentials. |
| Response Metric | Ensure that the Response metric category has only one numeric metric called Status. |
| | For more information, see Defining the Basic Response Metric Group. |
| Metric Definitions | Check that there are no hardcoded paths to Perl in your metric definitions. |
| Metric Definitions | Ensure that no defined key column stores Timestamp or Date or has key parts that are variable such as Date, Timestamp, Line Number, or Session Id. |
| Metric Definitions | Ensure that no credential values are passed as command-line arguments to scripts in the metric definitions |
| Metric Definitions | Ensure that user names and passwords are passed by stdin to scripts and not by environment variables. |
| Metric Collection Item | Do not use the UPLOAD_ON_FETCH attribute when defining a collection item. |
| | For more information, see Table 3-5 |

**Table B-2    (Cont.) Targets Checklist**

| Category | Checklist Item |
|---|---|
| Metric Definitions | Ensure that the display names for metrics and metric columns are user-friendly and have proper NLSID. For more information, see Metric Definition Files. |
| Metric Definitions | Ensure that configuration metric definitions use type RAW (not TABLE). For more information, see Table 3-4 |
| Metric Definitions | Ensure that any count or number type metric column is not defined as type STRING. |
| Metric Definitions | Ensure that metric keys do not have high cardinality, that is, a metric does not collect thousands of keys. For more information, see Defining Target Metadata |
| Metric Definitions | Ensure that you categorized your metrics within the Default metric class. For more information, see Categorizing Metrics. |
| Metric Definitions | Ensure that configuration metrics only collect data that is explicitly changed due to administrator actions. |
| Target Configuration Data | Ensure that the integer VER attribute is specified in the Configuration Metadata XML file. For more information, see Table 7-1 |
| Target Configuration Data | Ensure that table names begin with the plug-in tag followed by an underscore and can be a maximum size of 25 characters. |
| Target Configuration Data | Ensure that table and column names are in uppercase. |
| Target Configuration Data | Ensure that the size of columns is reasonable. |
| Target Configuration Data | Ensure that User Interface (UI) names for tables and columns are reasonable and user-friendly because these appear in the UI and can be seen by end users. |
| Target Configuration Data | Ensure that key columns are correctly identified for each table (for non-single row tables) |
| Target Configuration Data | Ensure that flags with default settings are not repeated. Do not repeat flags in every table or column that have default settings. Use flags for non-default setting overrides only. List all flags (including default setting) at METADATA level if you want to list them in one place. |
| Target Configuration Data | Ensure that the META_VER attribute in the target collection files matches the corresponding META_VER attribute defined for the target type. |
| Metric Collection Items | Ensure that the Response CollectionItem defined for the Response metric has a frequent schedule. Oracle recommends a collection interval between 1 and 5 minutes. |
| Metric Collection Items | Ensure that conditions are not defined on key columns |
| Metric Collection Items | Ensure that values used with MetricColl elements are valid metric values in the target type metadata XML. |
| Metric Collection Items | Ensure that messages defined for conditions use substitution variables for threshold values instead of hardcoding values. |
| Metric Collection Items | Ensure that Alert messages include metric display names, metric values and the thresholds that caused the alerts. Ensure that the main alert is conveyed in first 80 chars of the message. |
| Upgrading Targets | When upgrading a target type from an earlier release, ensure that data column types are not modified |
| Upgrading Targets | When upgrading a target type from an earlier release, ensure that the key columns, order, data type, or number are not modified |

**Table B-2  (Cont.) Targets Checklist**

| Category | Checklist Item |
| --- | --- |
| Upgrading Targets | When upgrading a target type from an earlier release, ensure that the metric type is not modified (such as TABLE to RAW) |
| Upgrading Targets | When upgrading a target type from an earlier release, ensure that for RAW metrics, STORAGE_TABLE_NAME or STORAGE_COLUMN_NAME are not modified |
| Upgrading Targets | When upgrading a target type from an earlier release, ensure that the USAGE_TYPE of a metric is not modified. |
| Derived Associations | Ensure that your derived association rules start with proper prefix. For more information, see Using Association Derivation Rules Syntax and Semantics |
| Derived Associations | Ensure that your triggers satisfy all trigger patterns in the guide. For more information, see About Regular Query and Trigger Patterns. |
| Derived Associations | Ensure that your rule query only contains simple joins and one FROM clause. (If it is more complex, then explain how performance will be ensured) |
| Derived Associations | Test performance for each perspective on which your rule might get triggered |
| Derived Associations | Check that you have necessary indexes defined for joined columns (especially for larger data tables) |

# Checking Customized UIs

Table B-3 provides a checklist which applies to the customized UI for your plug-in. This checklist is only applicable if you have a customized UI.

For more information about customized UIs, see Defining a Management User Interface .

**Table B-3  Customized User Interface Checklist**

| Category | Checklist Item |
| --- | --- |
| Adobe Flash Player | Check that your custom UI works with the supported version of Adobe Flash Player. See the Enterprise Manager certification matrix available on My Oracle Support for supported versions. https://support.oracle.com/ |
| Browser Version Compatibility | Check that your custom UI works with the supported versions of Web browsers. See the Enterprise Manager certification matrix available on My Oracle Support for supported versions. https://support.oracle.com/ |
| Accessibility | Ensure that the UI complies with the Oracle Global HTML Accessibility Guidelines (OGHAG). For more information, see Accessibility Guidelines. |
| Performance | Run performance tests on all new pages to ensure they load under a reasonable time limit. |

# Checking Job Types

Table B-4 provides a checklist which applies to job types defined for your plug-in. This checklist is only applicable if you have files in the *plugin_stage*/oms/metadata/jobTypes directory.

For more information about Job Types, see Adding Job Types .

**Table B-4    Job Types Checklist**

| Category | Checklist Item |
|---|---|
| Job Type Name | Ensure the job type name is of the form *plugin_tag_ VerbNoun*.<br>• By using the plugin_tag as the prefix, you ensure the job type does not conflict with similar named job types from other plug-ins<br>• An appropriate verb-noun combination ensures that the objective of the job type is clear to the customer.<br>Oracle recommends names such as oracle_as_RunHostProcess, oracle_db_BackupDB, oracle_as_RotateLogs |
| Display Names | Ensure the job type's NLS information clearly conveys the intention of the job type. Keep in mind that there might be similar named job types from other plug-ins.<br>Oracle recommends names such as dbBackupDB - "Backup Oracle Database", or asRotateLogs - "Rotate Weblogic Logs" |

# Checking Reports

Table B-5 provides a checklist which applies to reports defined for your plug-in. This checklist is only applicable if you have files in the *plugin_stage*/oms/metadata/reports directory.

For more information about reports, see Adding Information Publisher Reports and Developing BI Publisher Reports .

**Table B-5    Reports Checklist**

| Category | Checklist Item |
|---|---|
| SQL Usage<br>Information Publisher Reports only | Ensure that NamedSQL is used rather than a raw SQL statement parameter in a report.<br>Oracle recommends using Named SQL because it makes patching and changing your SQL more robust. If a user does a CREATE LIKE on your report, then they get an actual copy of the SQL statement unless you use Named SQL, in which case they get a pointer to the Named SQL. If you subsequently change the SQL, then the user gets the new copy from the Named SQL pointer. |
| BI Publisher Reports only | Ensure that the report includes proper header and footer subtemplates. |

# Testing your Plug-in

Table B-6 provides a checklist for self-testing your plug-in.

For more information, see Validating, Packaging, and Deploying the Plug-in .

**Table B-6    Plug-in Self-Test Checklist**

| Category | Checklist Item |
| --- | --- |
| Deployment Scenarios | Confirm that you tested in the following deployment sequence: |
| | 1. Deploy on Oracle Management Service (OMS) |
| | 2. Deploy on Management Agent |
| | 3. Remove from Management Agent |
| | 4. Remove from OMS |
| | 5. Redeploy on OMS |
| | 6. Redeploy on Management Agent |
| | For more information, see Importing and Deploying the Plug-in Archive into Enterprise Manager. |
| Upgrade | If this is not the first version of your plug-in, ensure that you have tested your plug-in upgrade on the same Enterprise Manager installations as the earlier supported versions of your plug-in. |

# C

# Metric Unit Standardization

Metric owners need to specify the metric unit and unit category for each of the metric column (except keys columns). The following table lists the standard metric units:

| Unit Category | Unit Code | Unit Display | Unit NLS ID |
|---|---|---|---|
| BOOLEAN | BOOLEAN | boolean | `EM_SYS_STANDARD_BOOLEAN_BOOLEAN` |
| COUNT | NA | n/a | `EM_SYS_STANDARD_COUNT_NA` |
| DATA_SIZE | BLOCK | blocks | `EM_SYS_STANDARD_DATASIZE_BLOCK` |
| DATA_SIZE | BYTE | Byte | `EM_SYS_STANDARD_DATASIZE_BYTE` |
| DATA_SIZE | KB | KB | `EM_SYS_STANDARD_DATASIZE_KB` |
| DATA_SIZE | MB | MB | `EM_SYS_STANDARD_DATASIZE_MB` |
| DATA_SIZE | GB | GB | `EM_SYS_STANDARD_DATASIZE_GB` |
| DATA_SIZE | TB | TB | `EM_SYS_STANDARD_DATASIZE_TB` |
| DATE | NA | n/a | `EM_SYS_STANDARD_DATE_NA` |
| ENUM | NA | n/a | `EM_SYS_STANDARD_ENUM_NA` |
| FREQUENCY | HZ | Hz | `EM_SYS_STANDARD_FREQUENCY_HZ` |
| FREQUENCY | KHZ | KHz | `EM_SYS_STANDARD_FREQUENCY_KHZ` |
| FREQUENCY | MHZ | Mhz | `EM_SYS_STANDARD_FREQUENCY_MHZ` |
| FREQUENCY | GHZ | GHz | `EM_SYS_STANDARD_FREQUENCY_GHZ` |
| FREQUENCY | THZ | MHz | `EM_SYS_STANDARD_FREQUENCY_THZ` |
| IPV4 | NA | n/a | `EM_SYS_STANDARD_IPV4_NA` |
| IPV6 | NA | n/a | `EM_SYS_STANDARD_IPV6_NA` |
| MESSAGE | NA | n/a | `EM_SYS_STANDARD_MESSAGE_NA` |
| NA | NA | n/a | `EM_SYS_STANDARD_NA_NA` |
| NAME | NA | n/a | `EM_SYS_STANDARD_NAME_NA` |
| PARAMETER | NA | n/a | `EM_SYS_STANDARD_PARAMETER_NA` |
| PERCENTAGE | PERCENTAGE | % | `EM_SYS_STANDARD_PERCENT_PERCENT` |
| PORT | NA | n/a | `EM_SYS_STANDARD_PORT_NA` |
| POWER | WATT | watt | `EM_SYS_STANDARD_POWER_WATT` |
| POWER | KILOWATT | kiloWatt | `EM_SYS_STANDARD_POWER_KILOWATT` |
| POWER | MEGAWATT | megawatt | `EM_SYS_STANDARD_POWER_MEGAWATT` |
| POWER | GIGAWATT | gigaWatt | `EM_SYS_STANDARD_POWER_GIGAWATT` |
| POWER | AMP | ampere | `EM_SYS_STANDARD_POWER_AMP` |
| PROPERTY | NA | n/a | `EM_SYS_STANDARD_PROPERTY_NA` |

| Unit Category | Unit Code | Unit Display | Unit NLS ID |
|---|---|---|---|
| RATE | ACCESSSEC | accesses per sec | EM_SYS_STANDARD_RATE_ACCESSPS |
| RATE | BLOCKSEC | blocks per sec | EM_SYS_STANDARD_RATE_BLOCKSEC |
| RATE | BPS | bytes per sec | EM_SYS_STANDARD_RATE_BPS |
| RATE | CENTISECONDSPR | centiseconds per request | EM_SYS_STANDARD_RATE_CENTISECONDSPR |
| RATE | CHARACTERSEC | characters per second | EM_SYS_STANDARD_RATE_CHARACTERSEC |
| RATE | CONNECTIONSEC | connections per second | EM_SYS_STANDARD_RATE_CONNECTIONSEC |
| RATE | GBPS | GB per sec | EM_SYS_STANDARD_RATE_GBPS |
| RATE | IOSEC | I/O per second | EM_SYS_STANDARD_RATE_IOPS |
| RATE | KBPS | KB per sec | EM_SYS_STANDARD_RATE_KBPS |
| RATE | MBPS | MB per sec | EM_SYS_STANDARD_RATE_MBPS |
| RATE | MESSAGEMINUTE | messages per minute | EM_SYS_STANDARD_RATE_MESSAGEPM |
| RATE | MESSAGESEC | messages per second | EM_SYS_STANDARD_RATE_MESSAGEPS |
| RATE | MILLISECREQUEST | ms per request | EM_SYS_STANDARD_RATE_MILLISECREQUEST |
| RATE | MINUTE | minute | EM_SYS_STANDARD_RATE_MINUTE |
| RATE | OPERATIONDAY | operations per day | EM_SYS_STANDARD_RATE_OPERATIONDAY |
| RATE | OPERATIONHOUR | operations per hour | EM_SYS_STANDARD_RATE_OPERATIONHOUR |
| RATE | OPERATIONINTERVAL | operations per interval | EM_SYS_STANDARD_RATE_OPERATIONINTERVAL |
| RATE | OPERATIONMINUTE | operations per minute | EM_SYS_STANDARD_RATE_OPERATIONMINUTE |
| RATE | OPERATIONSEC | operation per second | EM_SYS_STANDARD_RATE_OPERATIONSEC |
| RATE | OPERATIONTRANS | operations per transaction | EM_SYS_STANDARD_RATE_OPERATIONTRANS |
| RATE | REQUESTSEC | requests per second | EM_SYS_STANDARD_RATE_REQUESTPS |
| RATE | SECOND | per second | EM_SYS_STANDARD_RATE_SEC |
| RATE | TRANSACTIONSEC | transactions per second | EM_SYS_STANDARD_RATE_TRANPS |
| STATUS | NA | n/a | EM_SYS_STANDARD_STATUS_NA |
| TEMPERATURE | C | celsius | EM_SYS_STANDARD_TEMPERATURE_C |
| TEMPERATURE | F | fahrenheit | EM_SYS_STANDARD_TEMPERATURE_F |
| TIME | CENTISECONDS | centiseconds | EM_SYS_STANDARD_TIME_CENTISECONDS |

| Unit Category | Unit Code | Unit Display | Unit NLS ID |
|---|---|---|---|
| TIME | DAYS | days | `EM_SYS_STANDARD_TIME_DAYS` |
| TIME | HOURS | hours | `EM_SYS_STANDARD_TIME_HOURS` |
| TIME | MICROSEC | microseconds | `EM_SYS_STANDARD_TIME_MICROSEC` |
| TIME | MILLISECONDS | ms | `EM_SYS_STANDARD_TIME_MILLISEC` |
| TIME | MILLISECREQUEST | ms per request | `EM_SYS_STANDARD_TIME_MILLISECREQUEST` |
| TIME | MINUTES | minutes | `EM_SYS_STANDARD_TIME_MINUTE` |
| TIME | MONTHS | months | `EM_SYS_STANDARD_TIME_MONTHS` |
| TIME | NANOSEC | nanoseconds | `EM_SYS_STANDARD_TIME_NANOSEC` |
| TIME | SECOND | seconds | `EM_SYS_STANDARD_TIME_SEC` |
| TIME | SECONDREQUEST | seconds per request | `EM_SYS_STANDARD_TIME_SECONDREQUEST` |
| TIME | WEEKS | weeks | `EM_SYS_STANDARD_TIME_WEEKS` |
| TIME | YEARS | years | `EM_SYS_STANDARD_TIME_YEARS` |
| TIME_STAMP | NA | n/a | `EM_SYS_STANDARD_TIMESTAMP_NA` |
| URL | NA | n/a | `EM_SYS_STANDARD_URL_NA` |
| UTILIZATION | PERCENTAGE | % | `EM_SYS_STANDARD_UTILIZATION_PERCENTAGE` |

# Index

## D

## R

## S