Oracle® Communications Service Broker

System Administrator's Guide

Release 6.1

E29444-01

February 2013

ORACLE®

Oracle Communications Service Broker System Administrator's Guide, Release 6.1

E29444-01

Contents

8 Starting and Stopping the Administration Server

9 Setting Up Servers in Signaling and Processing Domains

10 Starting and Stopping Processing and Signaling Servers

11 Starting and Stopping the SS7 Process

12 Maintaining Oracle Communications Service Broker

13 Life Cycle of Processing Servers and Signaling Servers

14 Monitoring Service Broker Using Runtime MBeans

15 Tracing Sessions

Preface

This System Administrator's Guide describes how to perform many essential administrative tasks including: starting and stopping Oracle Communications Service Broker, adding and removing managed servers in a domain, configuring the Signaling Domain, configuring the Processing Domain, configuring the Orchestration Engine, and using scripts to automate procedures.

This guide also contains reference information in an appendix with details about directory structures, file content, environment variables, and system properties.

For a description of the system architecture, see *Oracle Communications Service Broker Concepts Guide.*

Audience

This guide is intended for system administrators, product integrators, and developers.

It assumes knowledge of the following subjects:

- How to deploy Service Broker and configure domains, managed servers, and other components
- The operator's network configuration
- Telephony networks and protocols
- Java programming and OSGi

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 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 Documents

For more information, see the following documents in the Oracle Communications Service Broker Release 6.1 documentation set:

- *Oracle Communications Service Broker Release Notes*

- *Oracle Communications Service Broker Concepts Guide*
- *Oracle Communications Service Broker Installation Guide*
- *Oracle Communications Service Broker Subscriber Store User's Guide*
- (Optional) *Oracle Communications Service Broker Online Mediation Controller Implementation Guide*
- (Optional) *Oracle Communications Service Broker Policy Controller Implementation Guide*
- *Oracle Communications Service Broker VPN Implementation Guide*
- *Oracle Communications Service Broker Social Voice Communicator Implementation Guide*
- *Oracle Communications Service Broker Security Guide*

Downloading Oracle Communications Documentation

Oracle Communications Service Broker documentation is available from the Oracle software delivery Web site:

http://edelivery.oracle.com/

Additional Oracle Communications documentation is available from Oracle Technology Network:

http://www.oracle.com/technetwork/index.html

1

About Domain Configuration

This chapter provides an overview of essential concepts and tools you need to understand before you can configure Oracle Communications Service Broker.

Configuration Overview

A Service Broker deployment can include two types of domains:

- Signaling Domain: Used to manage the of the Signaling Tier servers and the Signaling Server Units (SSUs) running on those servers.
- Processing Domain: Used to manage the Processing Tier servers and the Interworking Modules (IMs) and Orchestration Engine (OE) running on those servers.

When you use an Administration Console to update the domain configuration, your changes are implemented in the domain configuration directory. Each domain has an associated domain configuration that is stored in the domain configuration directory for that tier.

See the discusson on Service Broker deployments in *Service Broker Installation Guide* for more information.

About Domains

A Processing Domain contains a set of logically related Processing Servers, and a Signaling Domain contains a set of logically related Signaling Servers.

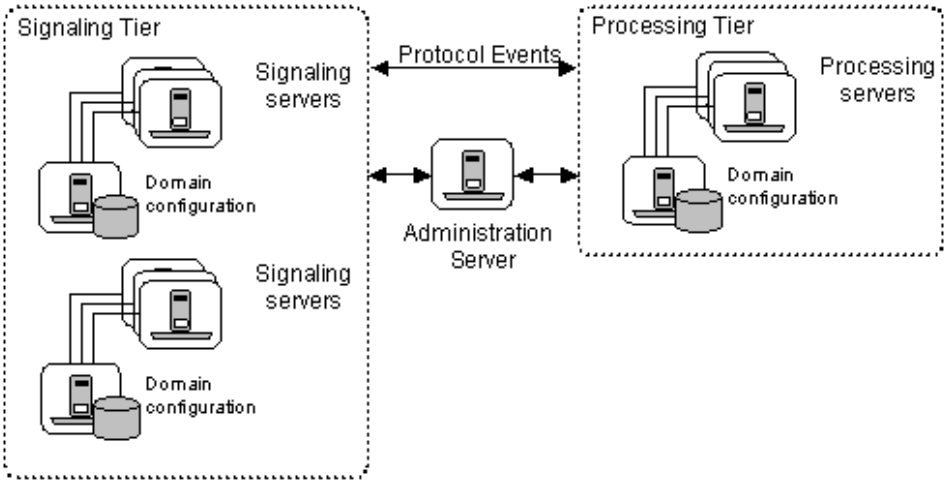
Managed servers in a given domain are symmetrical. All servers in the domain have the same software bundles deployed and started. Initial XML configuration data is stored inside the JAR file of the bundle that owns the configuration. Subsequent configuration changes are stored outside the software bundles in a separate location.

Configuration versioning is achieved by storing a copy of each version of the configuration JAR in an archive. At runtime, when the managed servers start only the most recent configuration version is pre-loaded into memory.

Managed servers have read-only access to the domain configuration that is stored in the domain configuration directory. Managed servers can access the domain configuration directory using either a shared file system or an HTTP/HTTPS connection to a domain Web server.

The domains interwork and propagate protocol events across tier boundaries. Figure 1–1 shows the tiers and the domains and how they work together.

Figure 1–1 Overview of Domains and Tiers



About Clusters

Service Broker creates managed server clusters using the Oracle Coherence technology which provides replicated and distributed data management and caching services on top of a highly scalable peer-to-peer clustering protocol.

Oracle Coherence automatically and transparently fails over and redistributes its clustered data management services. When a new managed server is added, or when a failed managed server is restarted, it automatically joins the cluster.

A Service Broker cluster also provides interprocess communication and persistence. Managed servers in all domains, including processing, signaling, and unified domains distribute events among each other across domain boundaries.

Locking the Domain Configuration for Changes

After you lock a domain configuration, you can make changes to it. When you lock a domain, it is locked for edits from all other administration clients. To apply the configuration changes you must commit them.

As long as you do not commit changes, you can discard them. After you discard all changes, the domain configuration is also unlocked.

Domain Configuration and Managed Server Update Modes

This section describes update options for making changes to the domain configuration and then propagating those configuration changes to the managed servers.

An update to the domain configuration and managed servers requires both of these steps:

1. "Updating the Configuration"
2. "Propagating Changes to the Managed Servers"

Updating the Configuration

After you lock a domain for editing, you can make changes to the configuration using either of the following modes.

- Autocommit mode

Autocommit is the default mode when using JMX-clients. Autocommit is not an option that is available when using the Administration Console.

When you update configurations using the Autocommit mode, your changes are committed and written to the configuration directory immediately. Each change is considered a separate transaction.

- Transaction mode

  When you update configuration in this mode, changes are not made in real time. Instead, multiple changes accumulate into one transaction. Transaction mode makes it possible to specify a set of configuration changes and have them applied all at once.

  Note: The Administration Server always uses the transaction mode.

You can manage transactions by using the following commands:

- Administration Console: Lock & Edit, Commit, Discard
- JMX-client: Begin, Commit, Rollback

Propagating Changes to the Managed Servers

The Domain Configuration Update Mode specifies how configuration updates are propagated to servers in the domain.

You can use either of the following modes:

- Online mode: Configuration updates are propagated to all servers in the domain when the changes are committed.
- Offline mode: Updates are carried out to the domain configuration and applied to each server only when the server is restarted.

  Setting the domain configuration offline makes it possible to perform a set of configuration updates and apply them the next time a server is restarted. This is used, for example, when either making configuration changes with no running servers (e.g. initial configuration), or when performing a rolling upgrade of an installation.

Configuration validation: The Service Broker validates configurations if made through either online or offline mode. An invalid configuration results in an error.

Accessing the Domain Configuration

The Service Broker Administration Server exposes a set of Java Management eXtensions (JMX) Configuration MBeans enabling JMX-compliant clients to manage and monitor the system.

After initial domain configuration, the typical method for a telecom carrier to work with Service Broker is to integrate it with a JMX-compatible Network Management System (NMS).

You can enable read and write access to the domain configuration using any of the following tools:

- Administration Console: A Web-based graphical user interface.
- JConsole: A component of the Java JDK located here: **`JDK_HOME/bin`**.

- Scripting Engine: The scripting format is an XML file that represent the MBeans, attributes and operations you want to configure. See "Using the Scripting Engine to Configure a Domain" for details.
- Other JMX-compliant clients: You can integrate a JMX-compatible Network Management System (NMS) and use it to configure and monitor Service Broker.

About the Administration Server

The Administration Server enables you to manage a domain configuration. The Administration Console displays the data stored in the domain configuration directory and provides write access.

Each Administration Server manages a single domain. In a typical production deployment you use two instances of the Administration Server: One for the signaling tier and another for the processing tier.

Note: In a test environment you can use a single unified domain that hosts one or more managed servers with each server performing the role of both the signaling and processing tiers. If you implement this type of test environment, you need to install only a single Administration Server.

You can display the Administration Console using a standard Web browser from any computer that has network access to the domain configuration directory. Configuration updates are propagated to all servers in the domain when the changes are committed.

Access to edit a domain configuration is available to one Administration Console at a time. All other users view the Administration Console in read-only mode though an administrator can seize the lock ("force lock") to go to edit mode.

The Administration Console provides users with intuitive application logic by transparently grouping Java objects into useful functional units. However, a site can integrate Service Broker with an NMS by connecting directly to the underlying Service Broker configuration and runtime MBeans.

Service Broker also provides remote monitoring using the Simple Network Management Protocol (SNMP). However, the Service Broker MIB objects are read-only. You cannot modify a Service Broker configuration using SNMP. See Chapter 16, "Remote Monitoring Service Broker with SNMP" for more information.

About Service Broker Configuration MBeans

Service Broker provides a standard software API to configure the Service Broker modules in the form of Java Management eXtensions (JMX) Configuration MBeans. JMX is a Java technology that provides tools to manage and monitor system resources.

The Service Broker Configuration MBeans provide an API to set/get configuration attributes and to perform operations for configuring the managed resources.

Each component in a domain exposes a set of MBeans, organized in a logical hierarchy, that together form the complete component configuration.

Using the configuration MBeans, Service Broker can be integrated with JMX-compliant clients such as a network management system, JConsole, or the Service Broker scripting engine.

Service Broker provides an MBeans Javadoc as part of its documentation. For more information, see the *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference*.

About JConsole

JConsole is a monitoring and management application that is included in the Java JDK. You can locate JConsole here: **JDK_HOME/bin/jconsole**.

You can use JConsole to do the following:

- View Service Broker configuration by viewing instances of configuration MBeans, as well as their attributes and operations
- Update Service Broker configuration by setting writable attribute values and invoking operations

About the Scripting Engine

You can manage and configure Service Broker programatically using scripts run by the Scripting Engine. Scripts can automate lengthy and recurring configuration tasks. For example, you can use a script to automate installing the Service Broker on multiple servers.

The scripting format is an XML representation of the MBeans. You define the MBean attributes to change and MBean operations to run and then execute the script from the Scripting Engine.

See "Using the Scripting Engine to Configure a Domain" for details.

2

Using the Administration Console to Configure a Domain

This chapter describes how to configure a Service Broker domain using the Administration Console.

About the Administration Console

The Administration Console provides a graphical user interface for performing configuration tasks easily and intuitively without programming knowledge. Each Administration Console manages a single domain.

The Administration Console should be used for the initial domain configuration. When your configuration changes are smaller, you can optionally use other clients.

After initial domain configuration, the typical method for a telecom carrier to work with Service Broker is to integrate it with a JMX-compatible Network Management System (NMS).

In a production deployment, usually you will deploy two instances of the Administration Server one to manage the Signaling Domain and another to manage the Processing Domain. Only a single Administration Server can be connected to a domain at a time.

To control points of failure you can do the following:

- Install each Administration Server on a separate physical server.
- Have available redundant Administration Servers that can be connected, only if needed, to a domain with a faulty server.

Audience For Using The Administration Console

The roles who use the Administration Console include the following: System Administrator, Service Operator, Network Engineer, and Support Engineer.

Tasks You Can Perform Using the Administration Console

You can perform many essential tasks using the Administration Console including the following:

- Add and configure managed servers in a domain
- Manage Signaling Domain Signaling Server Units (SSUs): Configure IP addresses, point codes, routes, timeout intervals
- Manage Processing Domain Interworking Modules (IMs) and supplementary IMs

- Configure the Orchestration Engine (OE)
- Manage Applications
- Manage the Subscriber Store
- Install and update software bundles
- Configure SNMP traps
- Configure Diameter AVPs
- Manage the credential stores
- Manage the persistent stores

Overview of the Administration Console User Interface

The Administration Console consists of several working areas described in Table 2–1.

Figure 2–1 Illustration of the Administration Console Working Areas

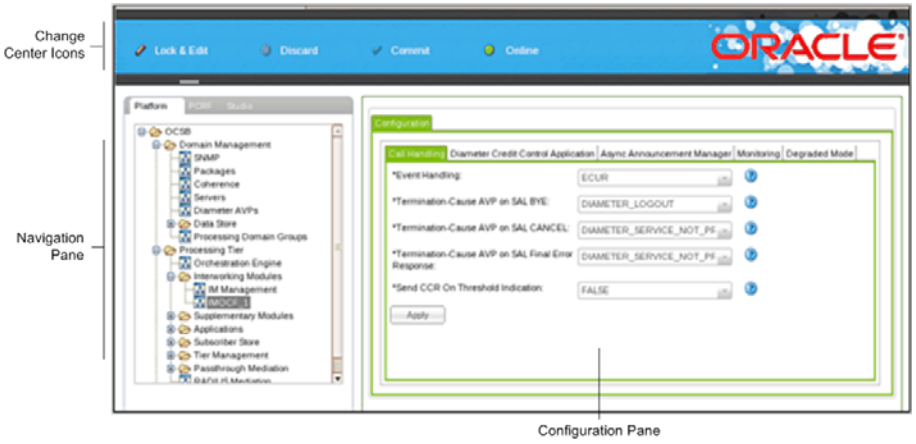


Table 2–1 Administration Console Working Areas

Working Area	Description
Navigation pane	Provides a logical tree-structure view of the domain. You can use it to access individual components deployed in the domain. In a Signaling Domain you can navigate through SSUs. In a Processing Domain you can navigate through the IM's, OE, Subscriber Store, and other components. The Navigation pane also includes Domain Management settings.
Configuration pane	When you select a component in the Navigation pane, configuration parameters appear in the right-side area. Use the Configuration pane to set up and modify the configuration of components deployed in the domain.
Change Center icons	A horizontal set of graphical icons you can use for managing a configuration change life-cycle. The change center operations include: Lock domain for changes, commit changes, and discard changes. See "Making Configuration Changes in a Domain" for more information.

Making Configuration Changes in a Domain

When you update the domain configuration using the Administration Console, Service Broker performs all changes in the transaction mode. This means that Service Broker accumulates multiple changes into a single transaction and applies all at once.

See "Domain Configuration and Managed Server Update Modes" for more information about the transaction mode.

You can perform the following actions when updating domain configuration:

- Lock Domain for Changes
- Commit Changes
- Discard Changes

Lock Domain for Changes

To make changes in the domain configuration, you must first lock the domain for changes. After you lock the domain, no one except you can make any changes to it. During the lock, Service Broker accumulates all configuration changes. Service Broker applies these changes only when you commit them.

To lock a domain for changes:

- In the Change Center pane, click the **Lock & Edit** button.

> **Note:** Access to edit a domain configuration is available to one Administration Console at a time. All other users view the Administration Console in read-only mode though an administrator can seize the lock ("force lock") to go to edit mode.

Commit Changes

You must commit your configuration changes for them to take effect. Service Broker applies all changes you made after locking the domain for changes.

To commit changes:

- In the Change Center pane, click the **Commit** button.

Discard Changes

As long as you do not commit changes, you can discard your changes. To discard changes, click the **Discard** button. After you discard all changes, the domain configuration is unlocked.

To discard changes:

- In the Change Center pane, click the **Discard** button.

Switching the Domain Configuration Update Mode

The Domain Configuration Update Mode specifies how configuration updates are propagated to servers in the domain. You can use one of the following modes:

- Online mode: Configuration updates are propagated to all servers in the domain when the changes are committed.
- Offline mode: Updates are carried out to the domain configuration and applied to each server only when the server is restarted.

Setting the domain configuration offline makes it possible to perform a set of configuration updates and apply them the next time a server is restarted. This is used, for example, when performing a rolling upgrade of an installation.

To switch between the Domain Configuration Update Modes:

- In the Change Center pane, click the **Switch to Offline Mode** button. To switch back from the offline to online mode, click this button again.

3

Using Service Broker MBeans to Configure a Domain

This chapter provides a high-level overview of using Service Broker MBeans to configure a domain.

About Service Broker MBeans

The Service Broker Administration Server exposes a set of Java Management eXtensions (JMX) Configuration MBeans enabling JMX-compliant clients to manage and monitor the system.

After initial domain configuration using the Administration Console, the typical method for a telecom carrier to work with Service Broker is to integrate it with a JMX-compatible Network Management System (NMS). The Service Broker configuration MBeans hierarchical tree-structure will be explained in this chapter using a standard JConsole MBeans client.

For more information see the following sources:

- *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference.*
- Service Broker runtime MBeans described in Chapter 14, "Monitoring Service Broker Using Runtime MBeans."
- Using SNMP to monitor Service Broker in Chapter 16, "Remote Monitoring Service Broker with SNMP."

Audience For Using Service Broker MBeans

The roles who use the Service Broker MBeans include the following: System Administrator, Integration Developer, Network Engineer, and Application Developer.

Tasks You Can Perform Using Service Broker MBeans

Using Service Broker MBeans, you can perform all tasks that are supported by the Administration Console including the following:

- Add and configure managed servers in a domain
- Manage Signaling Domain Signaling Server Units (SSUs): Configure IP addresses, point codes, routes, timeout intervals
- Manage Processing Domain Interworking Modules (IMs) and supplementary IMs
- Configure the Orchestration Engine (OE)

- Manage Applications
- Manage the Subscriber Store
- Install and update software bundles
- Configure SNMP traps
- Configure Diameter AVPs
- Manage the credential stores
- Manage the persistent store
- Configure a domain if the Administration Console cannot be accessed

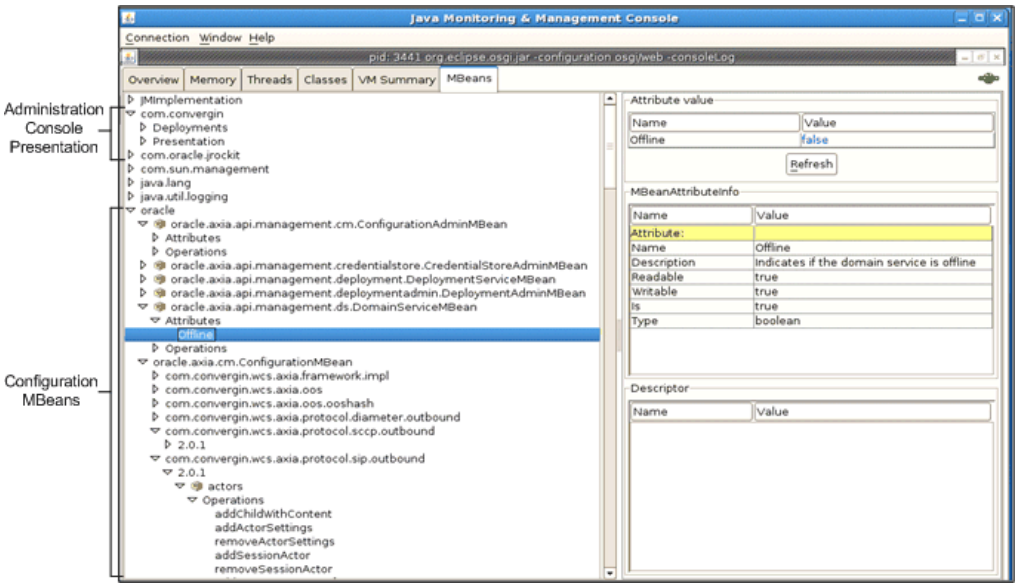
Overview of the Configuration MBeans Tree-Structure

This section describes at a high level the tree-structure of Service Broker configuration MBeans. For additional information, refer to the *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference*.

You access configuration MBeans by connecting your JMX-client to the Administration Console. If you connect to the managed servers, you will view runtime MBeans. See Chapter 14, "Monitoring Service Broker Using Runtime MBeans" for more information.

Do the following: Start JConsole, connect to the Administration Server process, and select the MBeans tab. You should see an MBeans tree-structure similar to this:

Figure 3–1 Service Broker Configuration MBeans Tree-Structure



Administration Server Deployments and Presentation MBeans

If you expand the **com.convergin** node, there are two nodes:

- Deployments: The MBeans under this node are used to add IMs via JMX.
- Presentation: The MBeans under this node are used to construct the Administration Console GUI. These MBeans are read-only.

Service Broker Configuration MBeans

The Service Broker configuration MBeans are located under the **Oracle** node. Expand the Oracle node, to view the Service Broker configuration MBeans. Most configuration MBeans are under the **oracle.axia.cm.ConfigurationMBean** node:

- oracle.axia.api.management.cm.ConfigurationAdminMBean
- oracle.axia.api.management.credentialstore.CredentialStoreAdminMBean
- oracle.axia.api.management.deployment.DeploymentServiceMBean
- oracle.axia.api.management.ds.DomainServiceMBean
- oracle.axia.api.management.upgrade.UpgradeManagementMBean
- oracle.axia.cm.ConfigurationMBean

Managing the Life Cycle of Domain Configuration Changes

To implement configuration changes you need to open and close the domain, lock and edit, commit changes, discard changes, and propagate the changes to managed servers.

> **Note:** Open domain and close domain operations are applicable only to JMX-clients and to scripts, not to the Administration Console.

Open a Domain for Configuration

To make changes in a domain configuration, you need to specify the path of the domain to open for configuration. After you open the domain for configuration, no one else can make any changes in the domain.

- Invoke the following operation of DomainServiceMBean:

```
void openDomain(String domainPath)
```

See the *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference* for more information.

Select a Domain Configuration Update Mode

After you open the domain for configuration, you can make changes to the configuration using either one of these modes:

> **Note:** These attributes control how changes to the configuration are made, not how the changes are propagated to the managed servers. See "Specify the Managed Servers Update Mode" for more information.

- Autocommit

  Autocommit is the default mode when using JMX-clients. Autocommit is not an option that is available when using the Administration Console.

  When you update configurations using the Autocommit mode, your changes are committed and written to the configuration directory immediately.

- Transaction

When you update configuration using this mode, multiple changes accumulate into one transaction. Using the transaction mode makes it possible to perform a set of configuration updates and then apply them all at once.

The read-only **TransactionActive** attribute of **ConfigurationAdminMBean** shows whether you are in transaction or autocommit mode.

To start a transaction:

- Invoke the following operation of **DomainServiceMBean**:

```
void begin()
```

To end a transaction:

- Invoke the following operation of **DomainServiceMBean**:

```
void commit()
```

Note: The DomainServiceMBean operations begin, commit, and rollback correspond to the Administration Console Lock&Edit, Commit, and Discard operations.

Close the Domain

To release the lock:

- Invoke the following operation of **DomainServiceMBean**:

```
void closeDomain()
```

See the *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference* for more information.

Specify the Managed Servers Update Mode

Use one of the following modes to specify how configuration changes are propagated to the managed servers:

- Online mode: Configuration updates are propagated to all servers in the domain when the changes are committed.
- Offline mode: Updates are carried out to the domain configuration and applied to each server only when the server is restarted.

  Setting the domain configuration offline makes it possible to perform a set of configuration updates and apply them the next time a server is restarted. This is used, for example, when implementing a rolling upgrade of a Service Broker installation.

To specify the Domain Configuration Update Mode:

- Set the **OffLine** attribute of **DomainServiceMBean** to one of the following values:
  - true
  - false

See "Domain Configuration and Managed Server Update Modes" for more information.

Manage the Domain Properties

Each domain configuration has a set of properties that are set during domain creation. The properties are defined as name-value pairs.

After the domain is created, you can change domain properties at any time using **DomainServiceMBean**.

To change a domain property:

- Invoke the following operation of DomainServiceMBean:

```
void setDomainProperty (name, value)
```

For more information, see the *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference*.

Naming Conventions For Service Broker Configuration MBeans

The Service Broker MBeans are registered in an MBean Server under a unique object name. The MBean instances reside in a hierarchy according to its object name. Each Mbean name enables a JMX-compliant client to retrieve an MBean of interest for a particular management operation.

Service Broker names its MBean objects as follows:

```
oracle:Type=oracle.axia.cm.ConfigurationMBean,Name=com.convergin<component-name.unique-resource>,Version=version_
number,name<index0>=resource,name<index1>=resource,name<index2>=resource
```

Table 3–1 describes the key properties that Service Broker encodes in its MBean object names.

Table 3–1 Service Broker MBean Object Name Key Properties

Property	Description
Type=*MBean-type-name*	The name of the MBean's type: oracle.axia.cm.ConfigurationMBean
Name=*component-name.unique-resource-name*	The name of the Service Broker component whose configuration is stored in the MBean, followed by a unique string that was provided upon creation of the MBean to identify the component resource which is represented by the MBean. The name is prefixed: com.convergin
Version=*version_number*	Specifies the version of the MBean instance. When you upgrade an MBean to a more recent version, this parameter enables Service Broker to keep the same name for different versions of the same MBean and use the version number to differentiate between them.
Name<index>=*resource*	Identifies object name segments using indexed property names.

For example this is the object name of a SIP protocol adapter MBean:

```
oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.protocol.sip.adapter,version=6.1.0,name0=ProtocolAdapter
```

Example: Configuring the SS7 Signaling Server Unit for SIGTRAN

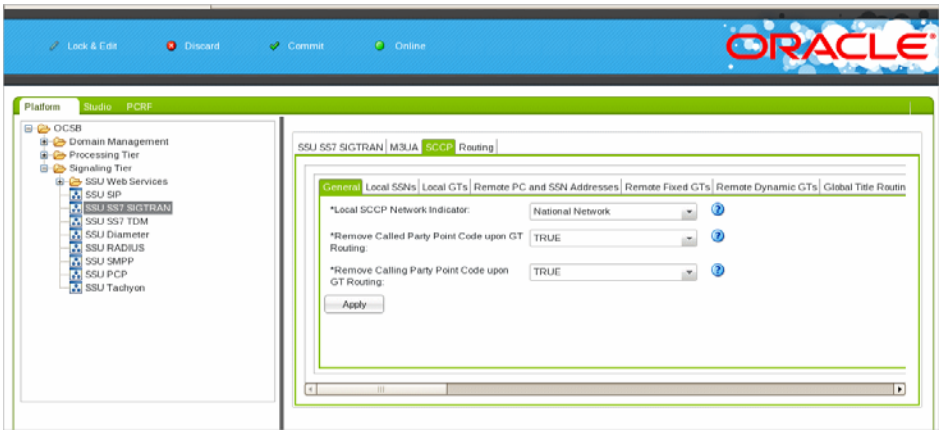
This section compares configuring an SS7 Signaling Server Unit for SIGTRAN when using the Administration Console and when using Service Broker configuration MBeans.

See the discussion on configuring the SS7 signaling Server unit for SIGTRAN in *Oracle Communication Service Broker Signaling Server Units Configuration Guide* for details on performing this task using the Administration Console.

The Administration Console provides application level logic that hides the complexity of accessing and configuring the underlying MBeans layer.

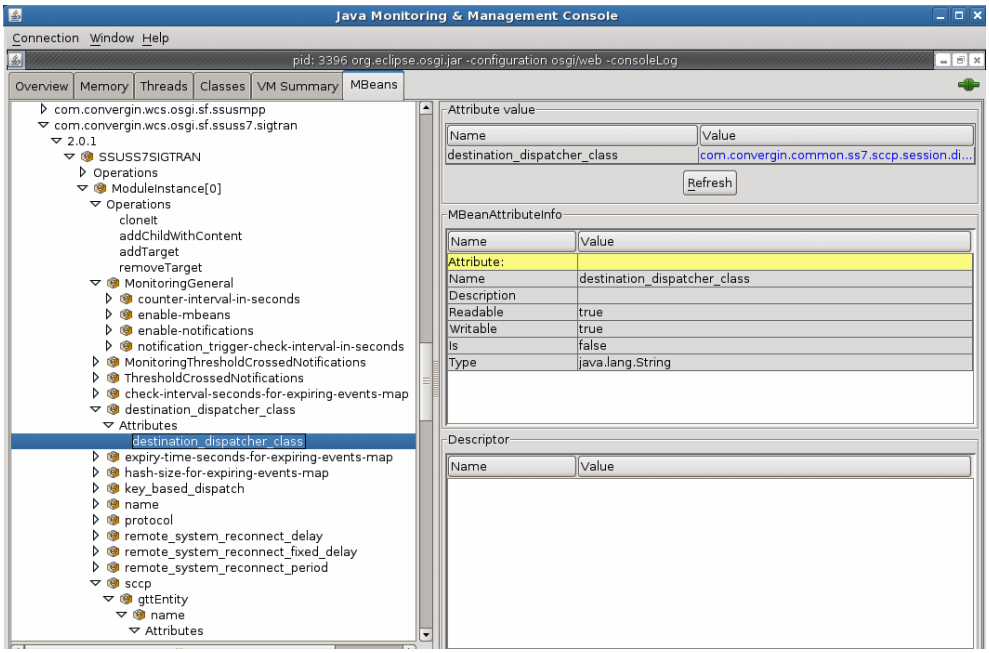
Figure 3–2 shows the Administration Console for inputting configuration values.

Figure 3–2 Administration Console: Configuring the SS7 Signaling Server Unit for SIGTRAN



To use MBeans to configure an SS7 Signaling Server Unit for SIGTRAN first start a JMX client such as JConsole. Figure 3–3 shows how you can use the JConsole to update Service Broker MBeans.

Figure 3–3 JConsole/MBeans: Configuring the SS7 Signaling Server Unit for SIGTRAN



Do the following:

1. Select the node **Oracle** then select **oracle.axia.cm.ConfigurationMbean**.
2. Select **com.convergin.wcs.osgi.sf.ssuss7.sigtran**. Each attribute and operation is represented by an MBean.
3. Follow the steps described in this section: "Managing the Life Cycle of Domain Configuration Changes".
4. Configure all relevant MBeans using the Administration Console to guide you through this task. See the discussion on configuring the SS7 signaling server unit for SIGTRAN in *Service Broker Signaling Server Units Configuration Guide* for details.

The Service Broker MBean layer is exposed primarily to enable a telecom carrier to integrate Service Broker with a JMX-compliant NMS.

Using a JConsole client for any configuration tasks is recommended only to facilitate the NMS integration or for any technical support inquiries.

4

Using the Scripting Engine to Configure a Domain

This chapter describes how to use the scripts to configure and manage a domain.

Scripts are used to automate configuration tasks you want to repeat multiple times. For example, you can use a script to configure multiple clusters of managed servers.

About Using Scripts

Processing Servers and Signaling Servers expose MBeans. The MBeans are accessible through the Scripting Engine.

The Scripting Engine itself has an MBean server that exposes MBeans related to configuration. These MBeans are accessible through the Scripting Engine.

The Scripting Engine executes operations on an MBean level. Operations and parameters are defined in a script that the Scripting Engine executes. The script format is an XML representation of the MBeans.

The script expresses the MBean operations you want to perform and all the data you want to provide as parameters to the operations.

Audience For Using The Scripting Engine

The roles who use the Scripting Engine include the following: System Administrator, Service Operator, Network Engineer, and Support Engineer.

Tasks You Can Perform Using the Scripting Engine

You can perform many essential tasks using the Administration Console including the following:

- Add and configure managed servers in a domain
- Manage Signaling Domain Signaling Server Units (SSUs): Configure IP addresses, point codes, routes, timeout intervals
- Manage Processing Domain Interworking Modules (IMs) and supplementary IMs
- Configure the Orchestration Engine (OE)
- Manage Applications
- Manage the Subscriber Store
- Install and update software bundles
- Configure SNMP traps

- Configure Diameter AVPs
- Manage the credential stores
- Manage the persistent stores
- Manage runtime operations

Script Syntax

A script has the top-level element *player*. Each operation to be performed is defined within the element *mbean*. Individual operations are defined within the element *operation*. Each parameter for an operation is defined within an element whose name is the same as the parameter name defined by the MBean.

Table 4–1 describes the syntax of the script file.

Table 4–1 Structure of an XML Management Script

Element	Description
player	Main element. Child element: mbean (zero or more) Optional attributes: ■ host ■ port This element defines which JMX server to connect to. The Scripting Engine and all Processing Servers and Signaling Servers have an MBean server. When updating configuration data, the Scripting Engine provides its own JMX server, so there is no need to specify a JMX server. The attribute **host** corresponds to the host name or IP address of the server where the Processing Server and Signaling Server are deployed. The attribute **port** corresponds to the JMX Registry port defined for the Processing Servers and Signaling Servers. The JMX Registry port for Processing Servers and Signaling Servers are defined in the domain configuration.
mbean	Parent element: player Child element: operation (zero or more) Attribute: name This element defines the object name of the MBean to use. The attribute **name** corresponds to the MBean class name. The fully classified name must be used. See the *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference* for information on MBean class names.
operation	Parent element: mbean Child element: *parameter_name* (zero or more) Attribute: name This element defines the operation to invoke on the Mbean defined in the element **mbean**. The attribute **name** corresponds to the name of the operation defined by the MBean.

Table 4–1 (Cont.) Structure of an XML Management Script

Element	Description
parameter_name	Parent element: operation or another *parameter_name* Child element: another *parameter_name* (zero or more) Attribute: no attribute For simple data types, the name of this element corresponds to the name of the in-parameter for the operation. All values of simple data types are represented as the String representation of the value. Boolean values are represented as [TRUE, FALSE] or [1, 0]. See "Representing Complex Data Structures" for complex data types.
set	Parent element: mbean Child element: none Attributes: ■ name ■ value This element defines which MBean attribute to set and the value to set it to. The MBean is defined in the parent element **mbean** element. The attribute **name** defines the name of the MBean attribute. The attribute **value** defines the value to set.
get	Parent element: mbean Child element: none Attribute: name This element defines which MBean attribute to get. The MBean is defined in the parent **mbean** element. The attribute **name** defines the name of the MBean attribute.

Setting and Getting Attributes

An MBean attribute can be **set** and **get** using the accessor methods of the attribute.

You get the value by prefixing the attribute name with get or is.

If the accessor method for an attribute is **is***Attribute_name* or **get***Attribute_name*, use the element **get** in the script to get the value.

If the accessor method for an attribute is **set***Attribute_name*, use the element **set** in the script to get the value.

Example 4–1 describes how to get the attribute **StartLevel** in the MBean **oracle.axia.api.management.agent.ManagementAgentMBean**. The corresponding method on the MBean is **int getStartLevel()**.

Example 4–1 Getting an MBean Attribute

```
<mbean name="oracle:type=oracle.axia.api.management.agent.ManagementAgentMBean">
   <get name="StartLevel"/>
</mbean>
```

Example 4–2 describes how to set the attribute **UseWellKnownAddress** on the MBean **CoherenceConfigTypeMBean**. The MBean is retrieved using the object name:

oracle:name=oracle.axia.storage.provider.coherence,name0=coherenceConfiguration,name1=useWellKnownAddress,type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0.

Example 4–2 Setting an MBean Attribute

```
<mbean
name="oracle:name=oracle.axia.storage.provider.coherence,name0=coherenceConfigurat
ion,name1=useWellKnownAddress,type=oracle.axia.cm.ConfigurationMBean,version=1.0.0
.0">
   <set name="UseWellKnownAddress" value="true"/>
</mbean>
```

Invoking Operations

MBean operations can be invoked from scripts.

You invoke the operation by defining the name of the operation and in-parameters.

Use the element **operation** to define that it is an MBean operation. Set the attribute **name** to the name of the operation. Define each in-parameter to the operations as an XML element and close the element **operation**.

Example 4–3 describes how to invoke the operation with the signature **void openDomain(java.lang.String domainPath)** with **domainPath** set to **/usr/local/sb/domain**.

Example 4–3 Invoking an MBean Operation

```
<operation name="openDomain">
  <domainPath>/usr/local/sb/domain</domainPath>
</operation>
```

Using Wildcard Characters in Scripts

The Scripting Engine supports the "*****" (asterisk) and "**?**" (question mark) wildcard characters to match MBean names. The asterisk matches any sequence of zero or more characters, and the question mark matches any single character.

This example matches all versions of this Diameter MBean.

```
oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.protocol.diameter,v
ersion=*
```

Creating and Using Variables in Scripts

Your Scripting Engine scripts gets the value of a variable using this notation:

${*variable_name***}**

Where:

variable_name is the name you give each variable.

You have these options for setting the value of a variable using the Windows-based operating systems:

- Using **-Dvar_name=***variable_name*.
- Using **result_property**.
- Entering a value manually at the script prompts (if the variable is undefined).

These options are explained in the following sections.

Using AXIA_OPTS to Create Variables

You define variables from the command line using the **AXIA_OPTS** environment variables.

The syntax for **AXIA_OPTS** is:

export AXIA_OPTS=-D*variable_name***=***variable_value*

For example, use this command to set the variable **domain.path** to **/domains/ocsb-basic-fs**:

export AXIA_OPTS=-Ddomain.path=/domains/ocsb-basic-fs

The value of **domain.path** is used in the script when the variable is referenced. This example references it:

```
<operation name="openDomain">
   <domainPath>${domain.path}</domainPath>
</operation>
```

AXIA_OPTS supports multiple arguments.

Using result_property to Create Variables

You can specify the Scripting Engine to capture the result of an MBean operations using the **result_property** attribute. This example

```
<mbean
name="oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.protocol.diam
eter,version=2.0.0,name0=ProtocolAdapter,name1=workManagers,name2=workManager[0],n
ame3=capacity">
  <operation name="capacity result_property="workmanager.capacity"/>
  <operation name="increaseWMcapacity>
    <arg1>${workmanager.capacity}</arg1>
  </operation
</mbean>
```

Entering Undefined Variable Values at a Script Prompt

If a variable is not defined in a script, the Scripting Engine prompts for a value of the variable at the command line. For example, if the variable **domain.path** is not defined as a system property and it is used in a script, the Scripting Engine prompts you for the value like this:

```
Enter a value for parameter 'domain.path':
```

Representing Complex Data Structures

You can use complex data structures as in-parameters to operations. When referring to elements of complex data structures in Java, the elements are normally addressed using dot-notation to address individual data structures in the tree. The XML representation uses elements to separate individual member variables until a simple data object is reached.

The XML representation of this type is derived from the Java class using reflection.

For example, consider the Java class:

```
public class AnAddress {
   String host;
   int port;
```

```
}
```

An object of this class is used as a parameter to the MBean operation.

When defining the object in Java it could look like this:

```
...
AnAddress anAddress = new AnAddress;
anAddress.host="localhost";
anAddress.port=9002;
...
```

The XML representation of the definition is

```
<anAddress>
        <host>localhost</host>
        <port>9002</port>
</anAddress>
```

Example Script

Example 4–4 illustrates a script that creates the domain configuration directory **/mydomain**, and defines the Processing Server **pn_2**. The server is defined to execute on **localhost**, use the administration port **8902**, listen to port **9002**, use the JMX port **10103**, and the JMX registry port **10003**. Finally, the script closes the domain.

Example 4–4 Script That Creates a Domain Configuration and adds a Processing Server.

```
<!-- player connects to a particular host and port -->
<player>
  <!-- one or more mbeans -->
  <mbean name="DomainServiceMBean">
    <operation name="createDomain">
      <domainPath>/mydomain</domainPath>
    </operation>
    <operation name="openDomain">
      <domainPath>/mydomain</domainPath>
    </operation>
    <operation name="addManagedServer">
      <name>pn_2</name>
      <host>localhost</host>
      <port>9002</port>
      <adminPort>8902</adminPort>
      <jmxJrmpPort>10103</jmxJrmpPort>
      <jmxRegistryPort>10003</jmxRegistryPort>
    </operation>
    <operation name="closeDomain"/>

  </mbean>
</player>
```

Starting the Scripting Engine

The Scripting Engine can be used for changing configurations and to monitor Processing Servers and Signaling Servers.

You start the Scripting Engine from the directory *Oracle_home***/ocsb61/admin_server**.

Oracle_home is the Oracle home directory defined when you installed Service Broker.

The Scripting Engine is invoked using the script:

./script.sh *xml_script_file*

Replace *xml_script_file* with the file name of to script you want to execute.

The process that starts the Scripting Engine must have read/write privileges on the file system where the domain configuration resides.

5

Managing the Domain Web Server

This chapter describes how to manage the domain web server.

About the Domain Web Server

After you installed Service Broker, when you ran the domain creation script you selected how to enable the managed servers to access the domain configuration directory. You selected to use either a 1) Web server in which case the domain is called a hosted domain, or to use a 2) shared file system which creates a nonhosted domain:

- A hosted domain contains a Web server that provides the managed servers in the domain either HTTP or HTTPS access to the domain configuration directory and the OSGi bundles.

  In a production environment be sure either enable HTTPS, or provide backup and redundancy for the Web server and domain configuration directory.

  Note: The Administration Server requires local file system access to the domain directory because it needs write access to configuration, bundles, credential store and domain lock files.

- Nonhosted domains connect to the domain configuration directory and the OSGi bundles using a shared file system not by using a Web server.

See *Oracle Communications Service Broker Release Security Guide* for more information about securing a domain.

If you created a hosted domain, you need to start the domain Web server.

Starting the Domain Web Server

The Domain web server must be started by a user who has read privileges to the Domain Configuration directory. See the discussion about configuring security between Service Broker components for more information about setting up user privileges.

To start the domain web server:

1. Open a command line shell.

   Note: You must be logged in as a user who has read privileges on the file system where the domain configuration resides.

2. Change the directory to:

   Oracle_home/**ocsb61/admin_server**

   Oracle_home is the Oracle home directory you defined when you installed the product.

3. Enter:

   ./host.sh *directory*

   Replace *directory* with the path to the domain configuration directory.

4. If HTTPS is enabled, enter the keystore password.

When the domain web server is started, servers can access the domain configuration.

The port is defined in the property **org.eclipse.equinox.http.jetty.http.port**. The default value is 9000. See "System Administrator's Reference" for more information.

Stopping the Domain Web Server

The recommended way to stop the domain Web server of the hosted domain is to kill the process in which it is running. Refer to the documentation of your operating system for more information.

6

Mapping Custom Server Names

This chapter describes how to map custom server names to names patterns required by Service Broker.

About Server Names

Processing Domain server names must use this format:

- For Signaling Servers: `ssu_`*server_number*. For example, the following names are valid: `ssu_1, ssu_2, ssu_3`.
- For Processing Servers: `pn_`*server_number*. For example, the following names are valid: `pn_1, pn_2, pn_3`.

During the installation, if you specified custom server names that do not follow these patterns, you need to map each custom server name to a name that follows the standard format as described above.

Mapping Custom Server Names to Service Broker Standard Names

You map server names using **ServersMBean**.

Figure 6–1 shows the **ServersMBean** hierarchy. ServersMBean might contain multiple instances of ServerMBean. Each instance of ServerMBean represents a single server whose name you want to map.

Figure 6–1 ServersMBean Hierarchy

To map a custom server name to a Service Broker standard name:

1. Create an instance of **ServerMBean** by invoking the following operation of **ServersMBean**:

   ```
   ObjectName createServer()
   ```

2. Set the **ManagedServerName** attribute of **ServerMBean** to the custom server name that you specified during the installation.
3. Set the **SbServerName** attribute of **ServerMBean** to a standard Service Broker name.

See "Servers MBean" and "Server MBean" for more information.

Servers MBean

ServersMBean is a root MBean for configuration of mapping between custom names that you specify for Signaling and Processing Servers and server names that follow the patterns required by Service Broker.

Factory Method

Created automatically.

Attributes

string Name
Specifies a name of the mapping configuration

int MaxServerNumber
Specifies a maximum number of servers whose names are to be mapped

Operations

ObjectName createServer()

Creates an instance of ServerMBean

void destroyServer()

Destroys an instance of ServerMBean

ObjectName[] getServer()

Gets an array of references to instances of ServerMBean

ObjectName lookupServer()

Returns a specified instance of ServerMBean

Server MBean

Using ServerMBean, you can map a custom name of one server to a server name which follows the pattern required by Service Broker.

Factory Method

Servers.createServer()

Attributes

string ManagedServerName
Specifies the custom server name that you specified during server installation.

string SbServerName
Specifies a name that follows the pattern required by Service Broker.

string SbServerId
Specifies a unique ID that the server uses when generating TCAP messages. The ID must be unique across all domains.

Operations

None

7

Configuring Coherence

This chapter describes how to configure IP addresses of Processing Servers and Signaling Servers to allow them to communicate with each other.

About Coherence

Using a technology called Oracle Coherence, Processing Servers and Signaling Servers distribute events across the domain boundaries.

To communicate with each other, servers need to be aware of each other's IP addresses. You can specify IP addresses of the servers using one of the following methods:

- IP multicast. In this case, you need to specify a single IP address. All servers use this address to send and receive broadcast messages.

  In addition to the multicast IP address, you can specify the maximum number of hops that an IP packet may traverse. This parameter is known as Time-To-Live (TTL).

  Service Broker uses IP multicast as the only option when you run the domain creation script.

  See "Setting Up IP Multicast" for more information.

- IP unicast. In this case, you need to specify the IP address of each server in all domains.

  See "Setting Up IP Unicast" for more information.

These two methods are mutually exclusive. You need to use one or the other in all your domains. The method that you should use depends on the configuration of your network, including topology, the number of servers, and possible restrictions imposed by your firewall or routers.

Setting Up IP Multicast

Because Service Broker uses IP multicast as the default option when you run the domain creation script, you need to specify IP multicast configuration parameters only when you want to modify the settings you defined during the domain creation process.

To set up IP multicast:

1. In the domain navigation pane, expand the **OCSB** node.
2. Expand the **Domain Management** node.

3. Select **Coherence**.
4. In the **General** tab, in the **useWellKnownAddress** list, select **FALSE**.
5. Click **Apply**.
6. Click the **Multicast** tab.
7. In the **MultiCastAddress** subtab, fill out the fields as follows:
   - In the **address** field, enter the multicast IP address. You can enter any value in the range from 224.0.0.0 to 239.255.255.255.
   - In the **port** field, enter the multicast port.
8. Click **Apply**.
9. Click the **TTL** subtab.
10. In the **TTL** field, specify the TTL. You can enter any value from 1 to 255.
11. Click **Apply**.
12. Stop all servers in all domains and then start them again. See "Starting and Stopping Processing and Signaling Servers" for more information.

Setting Up IP Unicast

To set up IP unicast:

1. In the domain navigation pane, expand the **OCSB** node.
2. Expand the **Domain Management** node.
3. Select **Coherence**.
4. In the **General** tab, in the **useWellKnownAddress** list, select **TRUE**.
5. Click **Apply**.
6. Click the **Unicast** tab.
7. In the **ServerName** subtab, click **New**.
8. In the **New: /Unicast/ServerName** window, in the **ServerName** field, enter a descriptive name of a Processing Server or Signaling Server. You use this name when specifying the IP address of the server.
9. Click **Apply**.
10. Click the **ServerAddress** subtab.
11. In the **Parent** list, select the server whose IP address you want to define. This list contains the servers that you previously defined in the **ServerName** subtab.
12. Click **New**.
13. In the **New: /Unicast/ServerAddress** window, fill out the fields as follows:
    - In the **address** field, enter the IP address of the server.
    - In the **port** address, enter the port of the server.
14. Click **Apply**.
15. Repeat this procedure for all Processing Servers and Signaling Servers.
16. Stop all servers in all domains and then start them again. See "Starting and Stopping Processing and Signaling Servers" for more information.

Configuring Cluster Node Death Detection Properties

Service Broker detects a cluster node death condition if there is a sustained node failure, network outage, or connection failure. If a node death is detected, it is considered lost from the cluster and a redundant server will automatically failover with no service loss.

The mechanism for node death detection is based on two Coherence properties specified in Table 7–1.

Coherence considers both the timeout and the retries count to decide if a node is dead. Using the default configuration, Coherence declares a node dead if two consecutive heartbeat requests time out. If the timeout property is set to the default five seconds, after ten seconds a node is considered disconnected from the main cluster.

You can change the property values in Table 7–1 because a ten-second duration might not be appropriate as the default time interval to trigger server failover for some deployments.

Table 7–1 Cluster Node Death Detection Properties

Property Name	Coherence Option	Service Broker default
tangosol.coherence.ipmonitor.pingtimeout	tcp-ring-listener/ip-timeout http://docs.oracle.com/cd/E15357_01/coh.360/e15723/appendix_operational.htm#BABBGCHI	5 (sec)
tangosol.coherence.ipmonitor.pingretries	tcp-ring-listener/ip-attempts http://docs.oracle.com/cd/E15357_01/coh.360/e15723/appendix_operational.htm#BABBGCHI	2

Automatic Server Shutdown

The following scenario will cause a managed server to automatically shut down:

1. A network failure causes the Coherence cluster to split up. While the cluster is split each part acts as an independent cluster.
2. The network problem is resolved and some node(s) can rejoin the main cluster. If the state of the separated nodes become inconsistent with the main cluster they will not be able to join.

To avoid the need to monitor the system and detect when the above situation has occurred, these actions take place: When a coherence cluster service restart occurs, the managed servers of the main cluster survive but the other managed server JVMs automatically shut down.

If the node is automatically shut down, you must restart the managed server unless you have some kind of process supervision implemented.

Data Cache Restart

After network recovery, as managed servers start to rejoin the cluster, nodes rejoining the cluster as non-senior members will have their caches restarted (that data is lost).

The following scenario enables ongoing calls and the data cache to be maintained:

You have deployed a processing domain and a signalling domain on machine_1 and another processing domain and a signalling domain on machine_2. If a network connection problem occurs that causes the machines to be split into two clusters, they

continue to handle traffic. Coherence keeps the backup copy of a cache entry on each machine. New sessions are handled successfully, and ongoing calls are processed successfully.

To return the cluster to a "normal state", one of the two sides must be restarted. If possible you should configure the associated load balancers to route new sessions to the side that will not be restarted. Then shut down that side shortly before restoring the network (if time of restoration is known).

When the network outage has been restored, start the managed servers that were shut down. During the network outage, the two sides can continue to run. If you know when the network will be restored, one of the sides should be shut down shortly before restoring the network.

8

Starting and Stopping the Administration Server

This chapter describes how to configure Service Broker using the Administration Servers.

Starting and Stopping the Administration Server

You can access the Administration Server using a standard web browser. The client is the Administration Console.

Starting the Administration Server

To start an Administration server:

1. Log in to the physical server where your software is installed.

   Note: You must be logged in as a user that has read and write privileges on the shared file system where the domain configuration for the installation you are going to use is stored.

2. Change the directory to:

   *Oracle_home***/ocsb61/admin_server**

   Oracle_home is the Oracle home directory you defined when you installed the product.

3. Enter:

   ./web.sh *Domain_configuration_directory*

   Domain_configuration_directory is the path to the domain configuration directory.

   Example:

   ./web.sh ../mydomain

4. When prompted for **Username** and **Password**, enter the authentication information you will use to authenticate for Administration Console sessions.

Logging In to the Administration Console

To log in to the Administration Console:

1. Open your Web browser.

2. Enter the URL:

   [https | http]://*host***:***port***/console**

   - **https** or **http** depends on your security configuration. See the discussion on enabling and disabling SSL in *Oracle Communications Service Broker Security Guide* for details on system security for details.
   - *host* is the IP-address or host name.
   - *port* is the port for the Administration Server. The default value for the port is 9000.

3. If it is the first time you are logging in with the Administration Console, you are prompted to accept the certificate provided in the keystore. This is done differently depending on which Web browser you use. It also depends on whether a self-signed certificate is used or the certificate was provided by a certificate authority.

4. When prompted, enter the user name and password.

   The authentication information must be identical to the information provided when the Administration Server was started.

Stopping the Administration Server

To stop the Administration Server, you can terminate the console in which it is running using the terminal interrupt command. On most systems, this command is mapped to the Control+c key sequence.

You can also stop the server by using the operating system **kill** command on the parent Java process for the server.

To use the **kill** command, you first identify the process ID of the Java container process for the Administration Server. You then pass the ID number to the **kill** command.

For example, on Linux:

```
$ ps -A | grep java
 3276 pts/2    00:00:50 java
$ kill -9 3276
```

About Setting Up Security for the Administration Server

The Administration Server supports authentication when connecting to the Processing Servers and the Signaling Servers. It also supports authenticates with a login screen when you use a browser to connect to the Administration Server.

See *Oracle Communications Service Broker Security Guide* for information about setting up security for the Administration Server.

9

Setting Up Servers in Signaling and Processing Domains

This chapter describes how to add servers to, and remove servers from, your Service Broker deployment using the Administration Console and Java MBeans.

Setting Up Servers Using the Administration Console

You can add and remove servers using the Server Configuration screen.

To access the Server Configuration screen:

1. In the domain navigation pane, expand **OCSB** and then expand **Domain Management**.
2. Select **Servers**.

Typing a server name into the **Filter** field displays a filtered list of servers.

Adding a Server to a Domain

Before you add a server to the Domain Configuration, you must install the Service Broker software on that server. See *Oracle Communications Service Broker Installation Guide* for instructions.

To add a server to a domain:

1. In the Servers List pane, click the **New** button.

   The Add Server dialog box appears.
2. Fill in the fields described in Table 9–1.
3. Click **Apply**.

Removing a Server from a Domain

Before you remove a server from the Domain Configuration you must stop the server.

To remove a server from the domain:

1. In the Servers screen, in the list of servers, select the check box corresponding to the server you want to remove.
2. Click **Delete**.

Setting Up Servers with Java MBeans

Using **DomainServiceMBean**, you can add servers to, and remove them from, a domain.

See the following sections for more information:

- Managing the Life Cycle of Domain Configuration Changes
- Example Script

Adding a Server to a Domain

When you add a server to a domain using Java MBeans, the process of adding the server requires the following:

1. Specifying properties of the server. For example, you need to define the name of a server, IP address of the system on which the server runs, and ports to be used in different states.
2. Setting up clustering parameters, if required
3. Setting up security for the server

To add a server to a domain:

1. Invoke the following operation of **DomainServiceMBean**:

   ```
   void addManagedServer(String name, String host, int port, int adminPort, int
   jmxJrmpPort, int jmxRegistryPort)
   ```

   Table 9–1 lists the parameters that you need to provide.

Table 9–1 Server Properties

Property	Description
name	The name of the server. The name must be unique across all domains. Format: alpha-numeric characters. Case-sensitive. No white spaces. Do not use white space in the name.
host	The host name or IP-address of the system where the server runs. Format: alpha-numeric. IP-address format or DNS name format.
port	This port setting is deprecated and no longer used. It should be set to its default value, -1.
adminPort	The IP port to use for the server when it is at SAFE level. This is the port used for configuration when the server is starting up. Format: numeric
jmxJrmpPort	The port to use for Java Remote Method Protocol (JRMP) invocations to the server. Format: numeric.
JmxRegistryPort	The port to use for the MBean Server on the server. Format: numeric.

2. If your Service Broker deployment uses well-known addresses to group the Processing Domain and the Signaling Domain, specify how the newly added

server distributes events to other servers across the domain boundaries by setting up clustering. See "Setting Up IP Unicast" for more information.

3. Configure security for the server. See *Oracle Communications Service Broker Security Guide* for more information.

Removing a Server from a Domain

When you remove a server from a domain using Java MBeans, the process of removing the server requires the following:

1. Stopping the server that you want to remove
2. Specifying the name of the server that you want to remove
3. Updating clustering parameters, if required
4. Uninstalling the software

To remove a server from a domain:

1. Stop the server that you want to remove by invoking the following operation of **ManagementAgentMBean**:

   ```
   void shutdown()
   ```

   See "ManagementAgentMBean" for more information about this MBean.

2. Invoke the following operation of **DomainServiceMBean**:

   ```
   void removeManagedServer(String name)
   ```

3. If your Service Broker deployment uses well-known addresses to group the Processing Domain and the Signaling Domain, remove it from the domain configuration. To remove a server from the Coherence configuration that uses well-known addresses to group domains, perform the following on all domains:

   a. Identify which **UniCastAddress** MBean corresponds to the server you are removing by checking the MBean attribute **ServerName**.

   b. Invoke the operation **removeUnicastAddress** on the MBean **UnicastAddressesMBean**.

4. Uninstall the software from the physical server using Oracle Universal Installer. See *Oracle Communications Service Broker Installation Guide* for information on uninstallation.

10

Starting and Stopping Processing and Signaling Servers

This chapter describes how to start and stop Processing and Signaling Servers.

Starting a Processing Server or a Signaling Server

After you add a server to a domain, you can start the server. If a domain contains several servers, you need to start each server individually.

To start a Processing Server or a Signaling Server:

1. Log in to the physical server where your server software is installed.
2. Change the directory to:

   *Oracle_home***/ocsb61/managed_server**

   Oracle_home is the Oracle home directory you defined when you installed the product.
3. Enter:

   ./start.sh *Server Domain_URI*

   - *Server* is the server name given when you configured your domain.
   - *Domain_URI* is the URI to the domain configuration. Always include **initial.zip**

     If your domain configuration is accessed using the Domain Web server, use the scheme **http://** or **https://** depending on the security settings.

     If your domain configuration is accessed using a shared file system, use the scheme **file://**.

   The following is an example of how the domain configuration is accessed using the Domain Web server:

   ./start.sh proc_srv_1 https://somewebserver.com:9000/initial.zip

   The following is an example of how the domain configuration is accessed using a shared file system:

   ./start.sh proc_srv_1 file://some_directory/mydomain/initial.zip

   In both cases the name of the server is **proc_srv_1**.
4. If you are using HTTPS, you are prompted for the keystore password.

Stopping a Processing Server or a Signaling Server

To stop a server, you can use one of the following methods:

- Using Java MBeans (see "Stopping a Server with Java MBeans")
- Using the Scripting Engine (see "Stopping a Server with the Scripting Engine")

Stopping a Server with Java MBeans

Using ManagementAgentMBean, you can stop a server.

To stop a server:

- Invoke the following operation of **ManagementAgentMBean**:

  ```
  void shutdown()
  ```

  See "ManagementAgentMBean" for more information about this MBean.

Stopping a Server with the Scripting Engine

You can use the Scripting Engine to invoke the **shutdown()** operation of ManagementAgentMBean that stops a specified server.

Example 10–1 shows a sample code that stops a server.

Example 10–1 Example Script for Stopping a Server

```
<player host="localhost" port="10003" registryPort="10103">
  <mbean name="oracle:type=oracle.axia.api.management.agent.ManagementAgentMBean">
    <operation name="shutdown"/>
  </mbean>
</player>
```

See "Using the Scripting Engine to Configure a Domain" for more information on the Scripting Engine.

11

Starting and Stopping the SS7 Process

This chapter explains how to start and stop the SS7 process on Signaling Servers.

About the SS7 Process

The SS7 process is a native SS7 stack wrapper, required in every solution in which Service Broker communicates with entities in the SS7 network. The SS7 SSU sends and receives SS7 traffic through the SS7 process.

The process is included in all Service Broker packages, excluding the Policy Controller that does not require SS7 connectivity.

The SS7 process needs to run on each of the servers where the SS7 SSU is deployed. In a clustered split domain (that is, when deploying separate Signaling and Processing domains), you run the SS7 process on each of the Signaling Servers in the Signaling domain. In a basic single domain deployment, you run the SS7 process on each of the servers in the domain.

There are two types of SS7 processes, one for each type of SS7 network: SIGTRAN and TDM. For each network type, there are two process binaries, one for Linux and another for Solaris.

The binaries of the SS7 process are located in the following directory:

*Oracle_home***/ocsb61/managed_server/ss7stack**

Starting the SS7 Process for SIGTRAN

To start the SS7 process for SIGTRAN networks:

- Depending on the operating system you use, enter one of the following in the command shell:
  - Linux: **convergin.ss7stack.sigtran.linux** *port_number*
  - Solaris: **convergin.ss7stack.sigtran.solaris** *port_number*

    port_number is the port that the SS7 process use to listen to messages from the SS7 SSU.

Starting the SS7 Process for TDM

To start the SS7 process for TDM networks:

- Depending on the operating system you use, enter one of the following in the command shell:

- Linux: **convergin.ss7stack.dialogic.linux** *port_number* **–src=***ss7_process_module_id* **-sccp** *dialogic_sccp_module_id*
- Solaris: **convergin.ss7stack.dialogic.solaris** *port_number* **–src=***ss7_process_module_id* **-sccp** *dialogic_sccp_module_id*

  port_number is the port that the SS7 process use to listen to messages from the SS7 SSU.

  ss7_process_module_id is the ID that the SS7 process use to identify itself when connecting to the underlying SS7 stack.

  dialogic_sccp_module_id is the identifier of the underlying SCCP module of the SS7 stack.

Stopping the SS7 Process

To stop the SS7 process, use the relevant commands of the operating system (for example, **kill**).

12

Maintaining Oracle Communications Service Broker

This chapter describes how you maintain your deployment and perform housekeeping.

Backing Up Your Installation

Backup can be done on a set of different levels:

- Processing Server and Signaling Server
- Administration client
- Domain configuration

Backing Up a Processing Server or a Signaling Server

To back up a Processing Server or a Signaling Server, back up all the files located in the following directory:

Oracle_home/**ocsb61/managed_server**

Oracle_home is the Oracle home directory you defined when you installed the product.

You should perform a backup immediately after installing, upgrading, or adding a patch to your Processing Server or Signaling Server.

See "Archiving and Cleaning Up Log Files" for information on backing up log files.

Backing Up an Administration Client

Backing up an administration client involves backing up the following:

- The Administration Server
- The Scripting Engine

To back up an administration client, back up all the files located in the following directory:

Oracle_home/**ocsb61/admin_server**

Oracle_home is the Oracle home directory you defined when you installed the product.

You should perform a backup immediately after installing, upgrading, or adding a patch to your administration client.

See "Archiving and Cleaning Up Log Files" for information on backing up log files.

Backing Up a Domain Configuration

To back up your domain configuration, back up all the files in your domain configuration directory. This directory was defined when you created the domain.

Perform backups on a regular basis and always immediately after:

- Updating or changing any configuration
- Adding or removing a Processing Server or Signaling Server from your installation
- Upgrading a Processing Server or Signaling Server
- Adding a patch to a Processing Server, Signaling Server, or an administration client

Backing Up Oracle Home

To back up your full Oracle home, back up all files and directories under *Oracle_home*.

Oracle_home is the Oracle home directory you defined when you installed Service Broker.

Processing Servers, Signaling Servers, and administration client installations, including log files, are also backed up when you back up *Oracle_home*.

Domain configurations are backed up if they are stored in an *Oracle_home* subdirectory.

Archiving and Cleaning Up Log Files

Log files are stored in the file system of your servers and administration clients.

You should archive and clean up your log files on a regular basis.

Log files for servers are by default stored directly under the directory:

*Oracle_home***/ocsb61/managed_server**

Oracle_home is the Oracle home directory you defined when you installed the product.

Log files for the administration clients are by default stored under the directory:

*Oracle_home***/ocsb61/admin_server**

Oracle_home is the Oracle home directory you defined when you installed the product.

Log files are stored using a roll-over pattern.

The file currently in use, *current_log_file* is named:

- **server.log** for Processing Servers and Signaling Servers
- **console.log** for administration clients

When *current_log_file* reaches a given size, the suffix **.***sequence_number* is added to the file name and a new *current_log_file* is created. The suffix **.***sequence_number* is a sequence number that is increased each time the file is rolled-over.

Log files with the suffix *sequence_number* can be archived for future reference and deleted. The roll-over occurs when the log file reaches a size of 100 KB.

The default log files are controlled by the configuration file named **log4j.xml** located in the directory:

- Processing Servers and Signaling Servers: **managed_server**
- Administration clients: **admin_server**

These directories are located under:

Linux and Solaris: *Oracle_home*/**ocsb61**

Oracle_home is the Oracle home directory you defined when you installed the product.

log4j.xml is a standard Log4J configuration file that can be changed to suit your needs. For detailed information on Log4J and the configuration file, see Log4J documentation at:

http://logging.apache.org/log4j/

About Patches and Patch Sets

A patch is always associated with a specific OSGi bundle, whereas a patch set is not associated with one specific bundle and might contain a collection of unrelated patches. A patch or patch set can be targeted to a specific bundle or to a server. The delivery format is ZIP files.

The patches and patch sets are either delivered with a bundled copy of Oracle Universal Installer or as a input file to be used with a standard installation of Oracle Universal Installer.

Instructions are shipped with the patch or patchset.

Managing Domain Bundles

The settings of each OSGi Bundle identify the bundle in the domain.

Table 12–1 describes OSGi Bundle properties.

Table 12–1 OSGi Bundle Properties

Property	Description
Name	Symbolic name of the OSGi bundle. Format: Alpha-numeric characters. Case sensitive. No spaces in the name.
Version	Version number of the bundle. Format: Alpha-numeric. IP-address form or DNS name format.
State	The state of the bundle: ■ Installed ■ Prepare Start ■ Start
Start Level	OSGi start level of the bundle Format: Numeric

The following sections describe how you can manage OSGi bundles with the Administration Console and Java MBeans.

- Managing Bundles with the Administration Console
- Managing Bundles with the DeploymentServiceMBean

Managing Bundles with the Administration Console

To access the Bundles Configuration screen:

1. In the domain navigation pane, expand **OCSB** and do one of the following:
   - Expand **Signaling Tier** and then expand **Domain Management**
   - Expand **Processing Tier** and then expand **Domain Management**
2. Select **Packages**.

The Packages configuration pane displays the properties described in Table 12–1.

Typing a package name into the **Filter** field displays a filtered list of packages.

Installing a Bundle

Before you install a bundle in the domain, you must extract a copy of the bundle in the Domain Configuration Directory.

To install a bundle:

1. In the Bundles screen, click **Install**.

   The Install dialog box is displayed.
2. In the **Location** column, type the location from where you extracted the bundle.
3. In the **Start Level** column, type a digit to indicate the level, then click **Apply**.

   The new OSGi Bundle now appears in the Bundle list.

Uninstalling a Bundle

Before you uninstall a bundle, you must stop the bundle. See "Stopping a Bundle" for instructions.

To uninstall a bundle:

1. From the Bundle list, select the check box corresponding to the bundle you want to uninstall.
2. Click **Uninstall**.

   The selected OSGi Bundle is removed from the list. The bundle is not deleted from the Configuration Directory.

Starting a Bundle

To start a bundle:

1. In the Bundle list, select the check box corresponding to the bundle you want to start.
2. Click **Start**.

Stopping a Bundle

To stop a bundle:

1. In the Bundle list, select the check box corresponding to the bundle you want to stop.
2. Click **Stop**.

Managing Bundles with the DeploymentServiceMBean

You can manage OSGi Bundles through JMX using the **DeploymentServiceMBean**, which exposes operations for installing and deploying these bundles.

13

Life Cycle of Processing Servers and Signaling Servers

This chapter describes how to manage the life cycle of Processing Servers and Signaling Servers.

Processing and Signaling Server Life Cycle

Oracle Communications Service Broker has life cycle states and transitions, as illustrated in Figure 13–1. The life cycle begins when Service Broker starts. OSGi defines a life cycle and Service Broker defines an overlay life cycle based on OSGi.

The life cycle is triggered when a server is started.

Figure 13–1 shows Service Broker states and transitions.

Figure 13–1 States and State Transitions

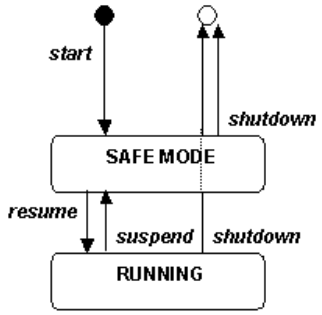


Table 13–1 describes the OSGi states and state transitions, and gives information on the corresponding OSGi start levels.

Table 13–1 OSGi Start Levels, Service Broker States, and State Transitions

State	Transition	Description
SHUTDOWN	N/A	Initial state. The server is not running. Corresponds to OSGi start level 0.

Table 13–1 (Cont.) OSGi Start Levels, Service Broker States, and State Transitions

State	Transition	Description
SHUTDOWN	*start*	This state transition is triggered by the server start script. Safe services are started. See "System Administrator's Reference" for details. Corresponds to OSGi start levels 0 to (SAFE MODE -1). If any error occurs during this transition, the OSGi framework shuts down.
SAFE MODE	N/A	The server can be managed on OSGi level and is running with a minimal set of OSGi bundles. Corresponds to OSGi start levels SAFE MODE to (RUNNING -1). Traffic is not processed and there is no interaction between Processing Servers and Signaling Servers.
SAFE MODE	*resume*	This state transition is triggered by the server start script and by management operations. The server transitions into state RUNNING.
SAFE MODE	*shutdown*	Two scenarios are possible: ▪ During server startup. That is, before entering state SAFE MODE from state initial. During this transition, the running OSGi bundles are shut down. Errors are logged. ▪ During server shutdown. That is, before entering state initial from state SAFE MODE. This state transition is triggered by management operations. If any error occurs during this transition, the transition is rolled back and the server retains state SAFE MODE.
RUNNING	N/A	When entering this state, all modules transition into state RUNNING.
RUNNING	*suspend*	This state transition is triggered by management operations and the server transitions into SAFE MODE.
RUNNING	*shutdown*	This state transition is triggered by management operations. During this transition, the server automatically continues with shutting down all running OSGi bundles.

Life Cycle Management MBeans

Life cycle management can be done using MBeans.The following sections provide reference information for the life cycle management of MBeans.

ManagementAgentMBean

Using ManagementAgentMBean, you can perform lifecycle management of a server though JMX. See "Processing and Signaling Server Life Cycle".

JAR File

oracle.axia.platform.managementagent-*version*.jar

where *version* is the version number of the JAR file: for example, **1.0.0.0**.

Package

oracle.axia.api.management.agent

Object Name

oracle:type=oracle.axia.api.management.agent.ManagementAgentMBean

Factory Method

Created automatically.

Attributes

int StartLevel
Read only.

Specifies the start level.

Operations

void forceShutdown()
Forces the server to transition to state SHUTDOWN.

void goToRunningLevel()
Transitions the server to state RUNNING.

void goToSafeLevel()
Transitions the server to state SAFE MODE.

boolean hasReachedSafeLevel()
Returns **true** if the server has reached start level SAFE MODE, otherwise **false**.

boolean serverIsRunning()
Returns **true** if the server has reached state RUNNING, otherwise **false**.

void shutdown()
Transitions the server to state SHUTDOWN.

14

Monitoring Service Broker Using Runtime MBeans

This chapter explains how to monitor Oracle Communications Service Broker.

Introduction to Service Broker Monitoring

You can monitor Service Broker using JMX Runtime MBeans which are simple Java objects that provide an API to:

- Poll values of Service Broker measurements, that is, counters, gauges and status
- Receive notifications when certain events occur

Runtime MBeans is a Java software API-based on the standard Java Management eXtensions (JMX). The API provides a system interface to monitor the activity of each Service Broker module.

Using JMX clients you can monitor any component (that is SSUs, IMs and SMs) in a Service Broker domain.

You access runtime MBeans by connecting your JMX-client to the managed servers process.

Understanding Service Broker Runtime MBeans

This section describes the Service Broker Runtime MBeans.

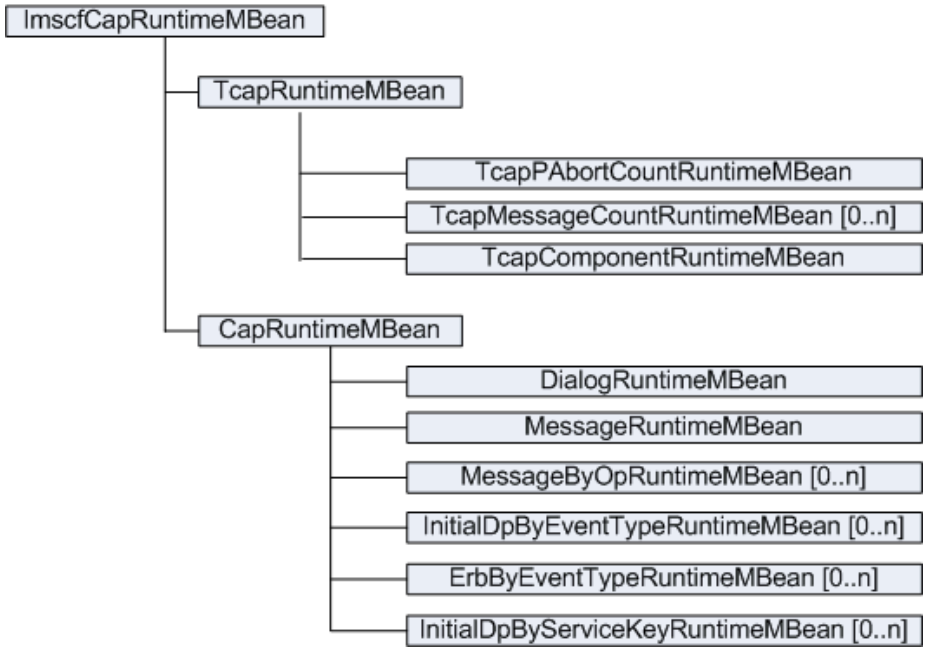
Service Broker Runtime MBeans Organization

Monitoring a Service Broker module involves polling measurements and statuses from a set of MBeans that together provide the module state. Each Service Broker module has a set of Runtime MBeans, which are organized in a hierarchy that includes:

- A root MBean that lets you monitor the functionality of the module. For example, the IM-SCF root MBean provides measurement on the number of sessions that the IM-SCF handled successfully or sessions that the IM-SCF handled unsuccessfully. The root MBean also provides references to other Runtime MBeans in the hierarchy.
- Other Runtime MBeans, each monitors a component or an interface of the module. For example, the IM-SCF TcapRuntimeMBean provides measurements regarding the TCAP interface.

Figure 14–1 shows an example of the Runtime MBean hierarchy for the IM-SCF CAP phase 1 Interworking Module. The root runtime MBean for M-SCF CAP phase 1 is ImscfCapRuntimeMBean. It contains TcapRuntimeMBean and CapRuntimeMBean.

Figure 14–1 Example of the IM-SCF CAP Runtime MBean Hierarchy



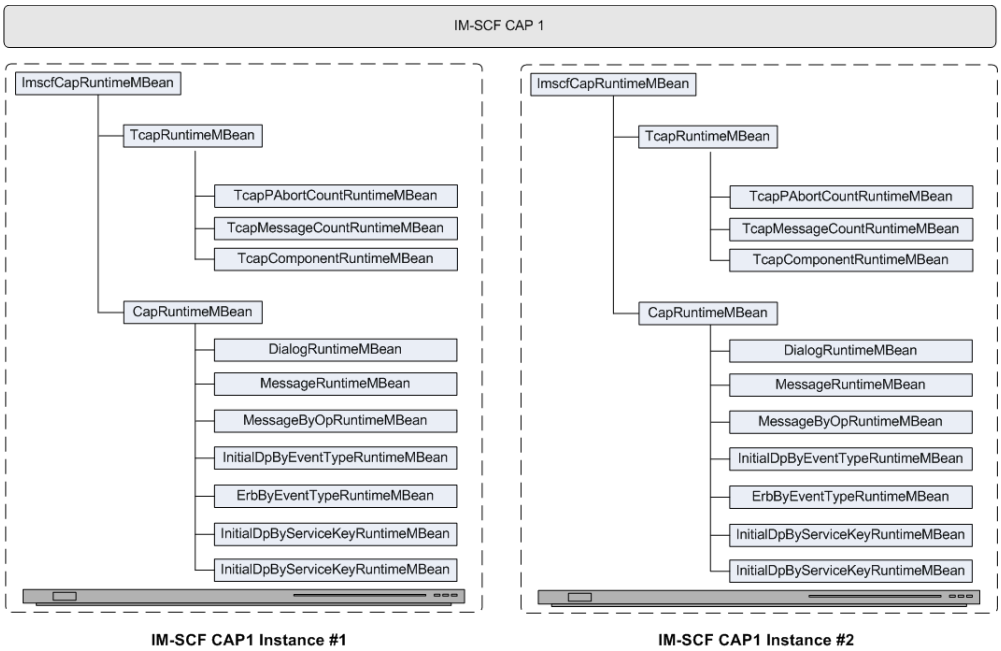
Each Runtime MBean provides reference to other Runtime MBeans under it in the hierarchy. Therefore, a JMX client can access a Runtime MBean by either directly looking it up in the MBean Server or by browsing down through the Runtime MBean hierarchy.

Service Broker Runtime MBean Instantiation

Each module instance instantiates its own Runtime MBeans on the server where the module instance is running. If Service Broker executes a module instance on more than one server, then the module instance will create several instances of its Runtime MBeans, one on each server. The Runtime MBeans on a server let you monitor the activity of the module instance running on that specific server.

Figure 14–2 shows an example of how IM-SCF instantiates Runtime MBeans on each of the servers on which IM-SCF is installed.

Figure 14–2 IM-SCF CAP1 Instances



Module instance Runtime MBeans are registered in the MBean server where the module instance is running. Runtime MBeans remain available in the MBean server as long as the module instance that instantiated them is running. When a module instance stops, its Runtime MBeans also cease to exist.

Service Broker Runtime MBean Object Names

Runtime MBeans are registered in the MBean Server under an object name of type javax.management.ObjectName. Service Broker naming conventions encode its Runtime MBean object names as follows:

`com.convergin:Type=`*`MBean-type-name`*`,Version=`*`version`*`,Location=`*`server-name`*`,Name=`*`module-instance-name.resource-name`*`,CountingMethod=`*`counting_method`*

Table 14–1 describes the key properties that Service Broker encodes in its Runtime MBean object names.

Table 14–1 Service Broker MBean Object Name Key Properties

Property	Description
Type=*MBean-type-name*	Specifies a short name of an MBean's type without the postfix MBean. For example, the Type parameter of LinksetRuntimeMBean is LinksetRuntime.
Location=*server-name*	Specifies a name of a Processing Server or Signaling Server on which the Runtime MBean runs.

Table 14–1 (Cont.) Service Broker MBean Object Name Key Properties

Property	Description
Version=*version_number*	Specifies the version of the containing bundle of the MBean instance. The ability to upgrade bundles is a key feature of OSGi containers. When you upgrade an MBean to a later version, this parameter enables Service Broker to keep the same name for different versions of the same MBean and use the version number to differentiate between them.
Name=*module-instance-name.resource-id*	Specifies the name of a Runtime MBean which consists of the following keys: ■ *monitor-instance-name*: Name of the module instance that the Runtime MBean monitors ■ *resource-name*: Resource that the Runtime MBean monitors. Possible values of this key depend on the type of MBean.
CountingMethod=*counting_method*	Specifies how an MBean counts events. You can set *counting_method* to one of the following: ■ CurrentInterval, when you want to get the number of times that a specific event occurred from the beginning of the time interval until the moment when you read the counter ■ PreviousInterval, when you want to get the number of times that a specific event occurred by the end of a previous time interval ■ CurrentGeneral, when you want to get the number of currently occurring events For more information on counting methods, see "Counters, Gauges, TPSs and Statuses".

For example:

```
com.convergin:Type=MessageByOpRuntime,Version=1.0.0,Location=sb_01,
Name=imscfcap4_instance.CAP.InitialDP,CountingMethod=CurrentInterval
```

Accessing Service Broker Runtime MBeans

Runtime MBeans are located and registered in the MBean Server of Processing Servers and Signaling Servers, where the Service Broker module instances are running.

Remote JMX clients (clients running in a different JVM than the Processing Server or Signaling Server), can use the javax.management.remote APIs to access any Service Broker MBean server. When accessed from a remote client, a Service Broker MBean Server returns its javax.management.MBeanServerConnection interface, which enables clients to access MBeans.

To monitor a module instance running on a Processing Server or a Signaling Server, a JMX client has to:

- Connect to the MBean Server on the server. This is where the Runtime MBeans are registered.

- Look up the Runtime MBeans in the MBean Server.
- Use the Runtime MBean interfaces to query counter and status values, and to receive notifications.

Runtime MBeans Reference

The Service Broker Runtime MBeans are described in details in JavaDoc. For more information, see the *Oracle Communications Service Broker Configuration and and Runtime MBean Java API Reference*.

Service Broker Measurements

This section describe various aspects of Service Broker measurements.

Counters, Gauges, TPSs and Statuses

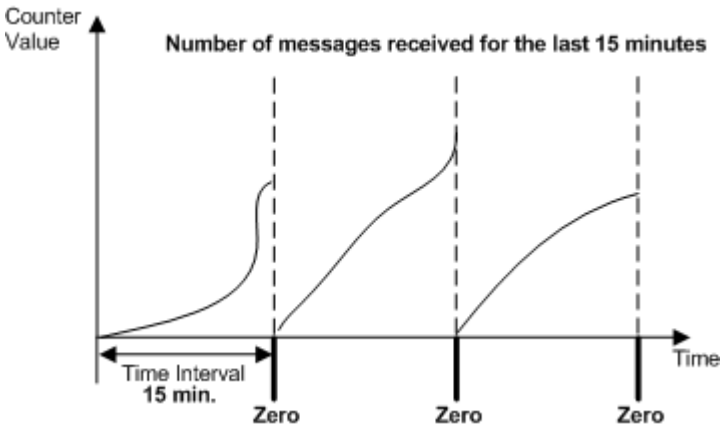
Service Broker Runtime MBeans provide the following types of attributes:

Counters

Counters store the number of times a particular event has occurred during the last time interval. For example, a counter can provide the number of messages received in the last 15 minute interval. The time interval is configured for each module instance depending on your specific needs. Whenever a time interval ends, Service Broker zeroes a counter.

Figure 14–3 shows how a counter increases and zeroes periodically in the end of each time interval.

Figure 14–3 Typical Counter Graph



Names of counter attributes have the **count** prefix.

Service Broker provides the following types of counters:

- Current interval counters

  This type of counter provides the number of times that a specific event occurred from the beginning of the time interval until the moment you read the counter. You can check how much time elapsed since the beginning of the current time interval using **WcsUptimeMBean**.

To make a counter act as a current interval counter, create an instance of an MBean and set the CountingMethod property of the MBean object name to CurrentInterval.

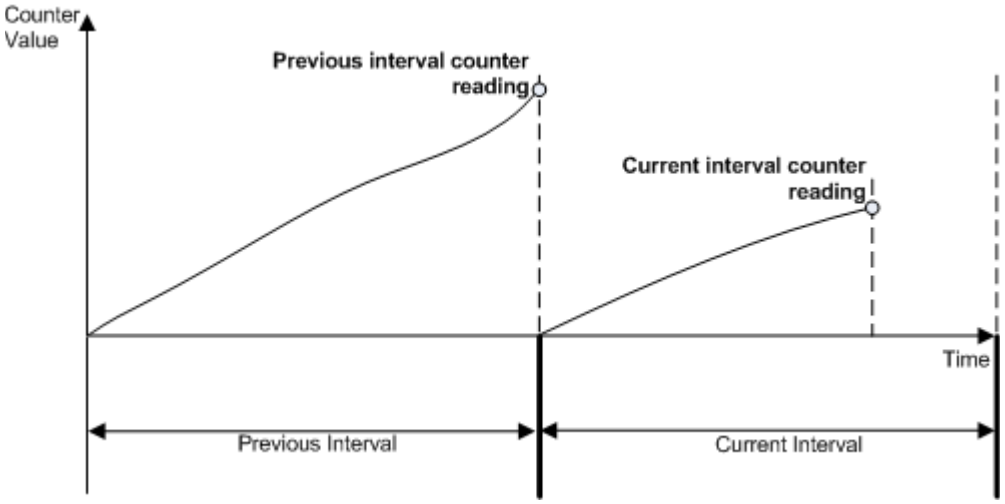
- Previous interval counters

  This type of counters provides the number of times that a specific event occurred by the end of a previous time interval.

  To make a counter act as a previous interval counter, create an instance of an MBean and set the CountingMethod property of the MBean object name to PreviousInterval.

Figure 14–4 shows an example of a current interval counter compared to a previous interval counter.

Figure 14–4 Current Interval Counter



Note: You need to access different instances of an MBean for each type of counter. The two MBean instances differ by the CountingMethod property of their object name. The value of the CountingMethod of one instance is CurrentInterval and of the second instance is PreviousInterval.

TPSs

TPSs are special counters whose time interval is non-configurable and preset to a very short duration of 15 seconds. There are only a few TPS counters. All of them have been identified as the most important Service Broker counters, which are critical to monitor Service Broker performance.

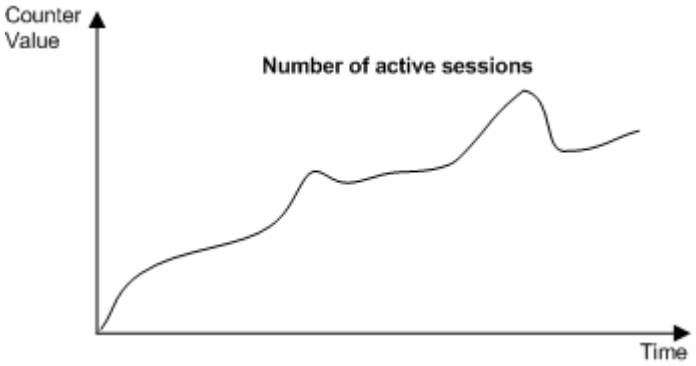
Names of TPS attributes have the **count** prefix. However, as opposed to regular counter attributes, TPS counters are available in MBean instances whose CountingMethod property is set to ShortInterval.

Gauges

Gauges store a number measured at a given moment. For example, the number of currently active sessions.

Figure 14–5 shows how a gauge value changes over time.

Figure 14–5 Typical Gauge Graph



Names of gauge attributes have the **gauge** prefix. Gauge attributes are available in MBean instances whose CountingMethod property equals CurrentGeneral.

Statuses

Statuses are attributes that indicate a state of a resource. For example, a status can provide information about a current state of an SS7 point code.

Module-Level Measurements and Tier-Level Measurements

Runtime MBeans can provide measurements described in "Service Broker Measurements" on the following levels:

- Module

  When a Service Broker module loads, Service Broker creates runtime MBeans for this module. Counters and gauges of runtime MBeans provide information about the module for which the MBeans were created. For example, when IM-SCF CAP1 loads, Service Broker creates runtime MBeans for monitoring the TCAP and CAP interfaces of the module. These MBeans contains counters and gauges that gather information about sessions and operations handled by IM-SCF CAP1.

- Tier

  To provide measurements on how an entire tier functions, Service Broker provides the runtime MBean that contains counters and gauges for gathering information related to the tier rather than to individual modules. For example, SystemCountersRuntimeMBean provides a counter that indicates the number of sessions opened in the last period in the Processing Tier.

  Table 14–2 describes attributes that enable you to monitor the Service Broker Processing Tier.

Table 14–2 Processing Titer Monitoring Attributes

Attribute	Description	Type	MBean
InitialRequestCount	Specifies the number of sessions opened in the Processing Server in the last measurement period	Counter	SystemCountersRuntime MBean
SessionGauge	Specifies the number of currently opened sessions in the Processing Server	Gauge	SystemGaugeRuntime MBean

Understanding Notifications

Service Broker notifications are based on Service Broker counters, gauges, TPSs and status attributes as described in "Service Broker Measurements". You can specify criteria for each Runtime MBean attribute (that is counter, gauge, TPS or status) that cause Service Broker to invoke a notification when each criteria is met.

In general, the notification mechanism involves the following steps:

1. An attribute value changes and reaches a criteria that was specified.
2. The Runtime MBean invokes a notification.
3. The attribute value changes again and the criteria is not met any longer.
4. The Runtime MBean clears the previously sent notification by invoking a cease notification.

Notifications are triggered by the Runtime MBean whose attribute is being monitored.

"Specifying Notification Criteria for Counters and TPSs" and "Specifying Notification Criteria for Gauges" explain how thresholds define reporting and clearance of a notification for counters and gauges.

Specifying Notification Criteria for Counters and TPSs

Counters store the number of times a particular event has occurred in the last time interval. For example, a counter can measure the number of InitialDP operations received in the last 15 minutes. See "Counters".

Table 14–3 explains how reaching an upper threshold or a lower threshold causes reporting and clearing of a notification.

Table 14–3 Reporting and Clearing Notifications for Counters

Threshold	Notification Sent When...	Notification Cleared When...
Upper	The counter crosses the upper threshold	The counter does not reach the upper threshold by the end of the last time interval
Lower	The counter does not cross the lower threshold by the end of the last time interval	The counter crosses up the lower threshold by the end of the last time interval

Specifying Notification Criteria for Gauges

Gauges are MBean attributes that provide a measurement at a given moment. For example, a gauge can contain a number of currently active sessions. See "Gauges".

Table 14–4 explains how reaching an upper threshold or a lower threshold causes reporting and clearing of a notification.

Table 14–4 Reporting and Clearing Notifications for Gauges

Threshold	Notification Sent When...	Notification Cleared When...
Upper	The increasing gauge crosses the upper threshold	The decreasing gauge crosses the threshold ceased value
Lower	The decreasing gauge crosses the lower threshold	The increasing gauge crosses the threshold ceased value

Specifying Notification Criteria for Statuses

Statuses are MBean attributes that indicate a state of a module. Service Broker triggers a notification when the value of an attribute changes to a certain value.

For example, SsuRemotePointCodeRuntimeMBean has the Status attribute that indicates availability of a point code. To monitor this attribute, you can define that Service Broker triggers a notification when the value of the Status attribute changes and indicates that the remote point becomes unavailable.

Table 14–5 explains how entering a specified state and exiting a state causes reporting and clearing of a notification.

Table 14–5 Reporting and Clearing Notifications for Statuses

Notification Sent When...	Notification Cleared When...
A monitored attribute changes to a specified state	A monitored attribute changes again to a state other than a specified state

Notification Structure

The Service Broker notification mechanism is based on javax.management.Notification and javax.management.AttributeChangeNotification Java classes.

These Java classes contain the following fields:

- Source

  The ObjectName of a Runtime MBean that sent the notification

- Sequence number

  This number allows individual notifications to be identified by a receiver

- Message

  Provides detailed textual description of a notification

- Type

  The type conveys the semantics of the notification in the following format: *rmb.attr.class*, where:

  – *rmb* defines a type of the runtime MBean that triggers a notification

  – *attr* defines an attribute of a runtime MBean whose change triggers the notification

  – *cause* defines a type of the change that triggers the notification. Table 14–6 describes notification causes that can be defined in the Type field.

Table 14–6 Notification Cause Values for Counters and Gauges

Cause	Description
highThresholdCrossed	A runtime MBean counter or gauge crosses a higher threshold.
clearedHighThreshold	The highThresholdCrossed notification was cleared.
lowThresholdCrossed	A runtime MBean counter or gauge crosses a lower threshold.
clearedLowThreshold	The lowThresholdCrossed notification was cleared.

Table 14–7 Notification Cause Values for State Change Indicators

Cause	Description
StateEntered	A runtime MBean attribute changed to a value.
StateCeased	A runtime MBean attribute is no longer equal to the value.

- Time

  Specifies the time when the event, which triggered the notification, occurred

- UserData

  Specifies additional data that the runtime MBean that triggers the notification wants to communicate.

  The UserData field can contain any data that a runtime MBean, which sends a notification, wants to communicate to a receiver. This field contains tag-value pairs separated by the semicolon.

  The following tags are allowed:

  - OriginNotificationSeqNum, which is used by a clearing notification to enable matching with the original notification that was cleared. In the clearing notification, this tag is set to the sequence number of the cleared notification. For more information, see "Receiving Notification Clearing".
  - LowThreshold, which is set to the crossed value of the lower threshold when the notification class set to **low** is triggered
  - HighThreshold, which is set to the crossed value of the upper threshold when the notification class is set to **high** is triggered
  - Match, which is set to the matched value when the notification class set to **match** is triggered
  - Differs, which is set to the value with which an attribute value was compared when the notification class set to **differs** is triggered

Receiving Notification Clearing

A runtime MBean clears a notification when criteria described in "Specifying Notification Criteria for Counters and TPSs" and "Specifying Notification Criteria for Gauges" are met.

To enable a notification receiver to recognize which original notification needs to be cleared, a runtime MBean uses the following methods:

- The OriginNotificationSeqNum tag in the UserData field of a clearing notification contains the sequence number of the original notification to be cleared.
- The Source field of the clearing notification contains the same value as the Source field of the original notification to be cleared.
- Clearing notifications contain the "ceased" postfix in the Type field. For example: Type=ImscfCapRuntimeMBean.SessionGauge.ceased.

Registering for Notifications

You can register for notifications using any JMX client. To receive notifications from several MBeans, you need to register for each MBean separately.

The following explains how to register for notifications using JConsole:

1. Start JConsole.
2. Click the **MBeans** tab.

   The MBeans tree view is displayed.
3. In the tree view pane, select the MBean for which you want to register.
4. Under the selected MBean, select **Notifications**.

   The Notification Buffer is displayed in the main pane.
5. Click **Subscribe**.

The registration created. When an event occurs, the notification is displayed in the Notification Buffer. You can clear the Notification Buffer at any time by clicking **Clear**.

Identifying Key Performance Indicators

Key performance indicators are measurements (counters, gauges, TPSs and statuses) that you monitor in order to assess the state and performance of your system.

Depending on your system and the modules installed in your system, you must identify the measurements that best serve the evaluation of your system. For more information about the measurements, see the discussion on monitoring Online Mediation Controller, Service Controller, and Policy Controller in the relevant implementation guides.

Use the Administration Console to configure the **Monitoring** tab of each module instance in your system. Define notifications that Service Broker will invoke, based on key performance indicators that you selected.

Set your NMS to monitor key performance indicators by either periodically polling the values of these measurements or register to notifications that you specified.

Configuring Service Broker Monitoring

You can configure monitoring separately for the following components: entire Processing Tier, each of the Processing Tier components, and Signaling Server Units (SSUs):

- Processing Tier:
  - Entire Processing Tier
  - Orchestration Engine (OE)
  - Interworking Modules
  - Supplementary Modules
- Signaling Tier:
  - SS7 SSU for SIGTRAN
  - SS7 SSU for TDM

Monitoring parameters that you can configure for each of these components are common for all the components. However, you can set up each of these parameters differently for different components depending on your specific requirements.

For example, you can define different triggers for generating notifications for IM-SCF and IM-SSF.

Warning: Thresholds that you define for notifications serve also as thresholds for key overload indicators. If you select a measurement (Runtime MBean and attribute) as a trigger for both notification and overload protection, the threshold value that you specify in the Monitoring tab is also regarded in the context of overload protection.

Accessing the Monitoring Configuration Screen

To access the Monitoring configuration pane:

1. Do one of the following according to the component you want to configure for monitoring:
   - Processing Tier: In the domain navigation pane, expand **OCSB**, expand **Processing Tier**, expand **Tier Management** and then select **Monitoring And Overload Protection**.
   - Orchestration Engine: In the domain navigation pane, expand **OCSB**, expand **Processing Tier**, and then select **Orchestration Engine**.
   - Interworking Module: In the domain navigation pane, expand **OCSB**, expand **Processing Tier**, expand **Interworking Modules** and then select the module you want to configure for monitoring.
   - Supplementary Module: In the domain navigation pane, expand **OCSB**, expand **Processing Tier**, expand **Supplementary Modules** and then select the module you want to configure for monitoring.
   - SS7 SSUs: In the domain navigation pane, expand **OCSB**, expand **Signaling Tier** and then select the SSU you want to configure for monitoring.
2. Click the **Monitoring** tab.

   Table 14–8 describes the subtabs in the Monitoring configuration pane.

Table 14–8 Monitoring Subtabs

Subtab	Description
General	Enables you to specify parameters that determine the general behavior of the notification mechanism. See "General" for more information.
State Changed Notifications	Enables you to configure generation of notifications when a value of an attribute of a specified Runtime MBean changes. See "State Changed Notifications" for more information.
Threshold Crossed Notifications	Enables you to configure generation of notifications when a threshold is passed. See "Threshold Crossed Notifications" for more information.

General

The General subtab enables you to specify parameters that determine a general behavior of the Runtime MBeans.

Table 14–9 describes configuration parameters in the General subtab.

Table 14–9 General Monitoring Parameters

Name	Type	Description
Enable Runtime MBeans	BOOL	Specifies whether JMX Runtime MBeans are enabled. Possible values: ■ TRUE ■ FALSE Default value: TRUE
Enable Notifications	BOOL	Specifies whether JMX notifications are enabled. Possible values: ■ TRUE ■ FALSE Default value: FALSE
Notification Sample Interval (sec)	INT	Specifies the interval between consecutive sampling of counters. Counters are sampled every few seconds to check whether to invoke notifications. Default value: 30

State Changed Notifications

The State Changed Notifications subtab enables you to configure a notification that Service Broker generates when an attribute of a specified Runtime MBean changes to a specified value.

The State Changed Notifications subtab contains a table in which each row represents an individual notification trigger condition based on an attribute of a specific Runtime MBean.

Table 14–10 describes configuration parameters in the State Changed Notifications tab.

Table 14–10 State Changed Notifications

Field	Type	Description
Name	STRING	Specifies a notification name
Enabled	BOOL	Specifies whether the notification is enabled. Possible values: ■ TRUE ■ FALSE Default value: TRUE
MBean Type	STRING	Specifies a type of the Runtime MBean to be monitored
MBean Attribute	STRING	Specifies an attribute of the MBean defined in the MBean Type field to be monitored. Notice that in this parameter, you can specify only status attributes. Status attributes indicate a state of a resource.
Value	STRING	Specifies an attribute value that triggers generation of the notification

Table 14–10 (Cont.) State Changed Notifications

Field	Type	Description
Server Filter	STRING	Specifies a filter that must be applied on the "server" key property on an MBean's instance name. If this parameter is defined, only the MBean instances with the registration name that matches the filter trigger the notification. The value of this parameter must be defined in the form of a regular expression. Default value: \S* (no filter)
Resource Filter	STRING	Specifies a filter that must be applied on the "name" key property of an MBean's instance name. If this parameter is defined, only the MBean instances with the registration name that matches the filter trigger the notification. The value of this parameter must be defined in the form of a regular expression. For example, to monitor MessageByOperationRuntimeMbean's counters of a specific operation, set ResourceFilter to "\S*InitialDP\S*". Default value: \S* (no filter)
Message	STRING	Specifies a message to be set in the "message" field of the notification. Default value: "notification"

Threshold Crossed Notifications

The Threshold Crossed Notifications subtab enables you to configure generation of notifications when a threshold is passed. The Threshold Crossed Notifications subtab contains a table in which each row represents an individual notification triggering condition based on an attribute of a specific Runtime MBean.

Table 14–11 describes configuration parameters in the Threshold Notifications tab.

Table 14–11 Threshold Crossed Notifications Fields

Field	Type	Description
Name	STRING	Specifies a unique notification name
Enabled	BOOL	Specifies whether the notification is enabled. Possible values: ■ TRUE ■ FALSE Default value: TRUE
MBean Type	STRING	Specifies a type of the Runtime MBean to be monitored
MBean Counting Method	STRING	Specifies the counting method. Possible values: ■ CurrentIntervalDeltaValue ■ PreviousIntervalDeltaValue ■ CurrentGeneralValue ■ PreviousShortIntervalDeltaValue Select CurrentIntervalDeltaValue.

Table 14–11 (Cont.) Threshold Crossed Notifications Fields

Field	Type	Description
MBean Attribute	STRING	Specifies an attribute of the MBean defined in the MBean Type field to be monitored
Class	STRING	Specifies a notification class. Possible values: ■ High A notification is triggered when a Runtime MBean counter or gauge crosses an upper threshold. ■ Low A notification is triggered when a Runtime MBean counter or gauge crosses an lower threshold. Default value: High
Threshold Value	INT	Specifies a high threshold, when the Class field is set to High, or low threshold, when the Class field is set to Low.
Threshold Ceased Value	INT	Specifies a threshold for which the "ceased" notification must be triggered
Server Filter	STRING	Specifies a filter that must be applied on the "server" key property on an MBean's instance name. If this parameter is defined, only the MBean instances with the registration name that matches the filter trigger the notification. The value of this parameter must be defined in the form of a regular expression. Default value: \S* (no filter)
Resource Filter	STRING	Specifies a filter that must be applied on the "name" key property of an MBean's instance name. If this parameter is defined, only the MBean instances with the registration name that matches the filter trigger the notification. The value of this parameter must be defined in the form of a regular expression. For example, to monitor MessageByOperationRuntimeMbean's counters of a specific operation, set ResourceFilter to "\S*InitialDP\S*". Default value: \S* (no filter)
Threshold Crossed Message	STRING	Specifies a text for the notification message that Service Broker generates when a threshold is crossed.
Threshold Ceased Message	STRING	Specifies a text for the notification message that Service Broker generates when a notification is ceased.

15

Tracing Sessions

This chapter describes how to trace individual sessions in Online Mediation Controller and Service Controller.

About Tracing Sessions

Using Service Broker, you can trace individual sessions. This feature is useful when you need to check how Service Broker components handle a specific session rather than reviewing an entire log of all sessions.

You specify the session to be traced by defining a string that Service Broker should search for in an initial event of the session. If Service Broker finds this string in the initial event, Service Broker traces the session.

The format for specifying the string and the locations within the message in which Service Broker searches the string depend on the message's protocol. For example, to trace a SIP session, you specify the user part of a SIP URI. Service Broker searches this SIP URI in the **P-Requested-Identity**, **To**, **From**, and **RequestURI** headings. Alternatively, for a Diameter session, Service Broker searches the specified string in the **Calling-Party-Address** AVP and **Subscription-id-data** AVPs.

Service Broker writes the traced information into the log file. All log files are stored in **/ocsb61/managed_server/**.

Service Broker stops tracing the session when the session is released.

About the Format of the Search String

The following sections explain the format in which you specify the string and the locations within the messages where Service Broker searches:

- SIP. See "SIP" for more information.
- SMPP. See "SMPP" for more information.
- CAP. See "CAP" for more information.
- WIN. See "WIN" for more information.
- AIN. See "AIN" for more information.
- MAP ANSI. See "MAP ANSI" for more information.
- Diameter. See "Diameter" for more information.

SIP

Service Broker searches in the following SIP headers:

- **P-Asserted-Identity**
- **From**
- **Request URI**
- **To**

To define a SIP URI, specify the user part of the SIP URI. To define a Tel URI, specify the telephone number.

SMPP

Service Broker searches in the following SMPP fields:

- **dest_addr**
- **source_addr**

In the search string, specify the telephone number of the call recipient or originator.

CAP

Service Broker searches in the following CAP headers:

- **Calling-party-number**
- **Additional-calling-party-number**
- **Called-party-number**
- **Called-party-BCD-number**
- **Destination-subscriber-number**

In the search string, specify the BCD digits.

WIN

Service Broker searches in the following WIN headers:

- **Mobile-Directory-Number**
- **Calling-Party-Number-Digits1**
- **MsId**
- **Digits**

In the search string, specify the BCD digits.

AIN

Service Broker searches in the following AIN headers:

- **Calling-Party-Id**

In the search string, specify the BCD digits.

MAP ANSI

Service Broker searches in the following MAP ANSI headers:

- **Mobile-Directory-Number**
- **MS-ID**

In the search string, specify the BCD digits.

Diameter

Service Broker searches in the following Diameter AVPs:

- **Service-Information AVP /Ims-Information AVP/Calling-Party-Address AVP** (Ro and Rf). In the search string, specify the full SIP URI of the calling party, including the SIP prefix, user part, and domain part. For example: **sip:calling@seagull.com**.
- **Subscription-id AVP/Subscription-id-data AVPs** (Ro only). In the search string, specify the identifier of the subscriber. For example: **imsiNumber**.
- **Service information/Subscription-id/Subscription-id-data AVPs** (Rf only). In the search string, specify the identifier of the subscriber. For example: **imsiNumber**.

About Session Tracing Modes

The user needs to select one of the following modes of tracing:

- Internal events tracing. See "Internal Events Tracing" for more information.
- Full session tracing. See "Full Session Tracing" for more information.

Internal Events Tracing

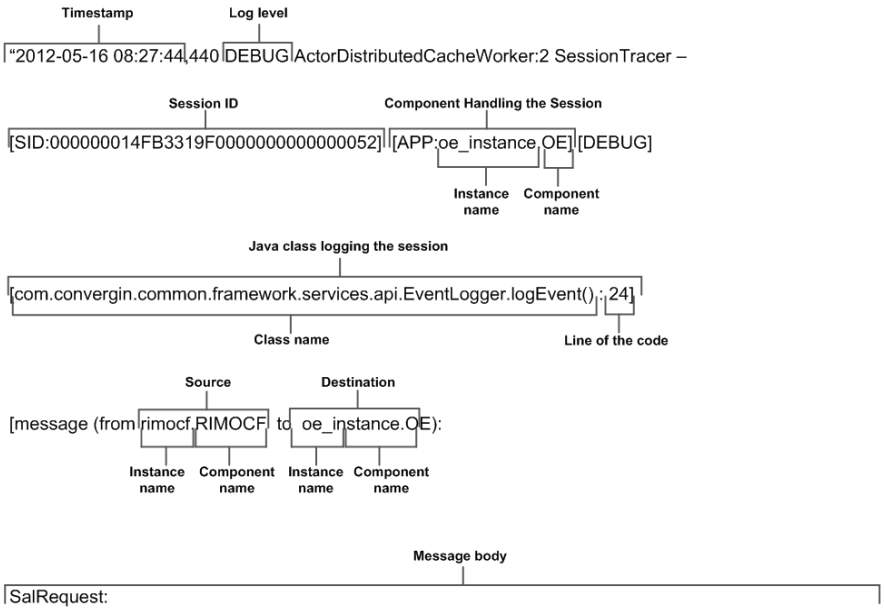
You might need to trace internal events when you want to monitor the behavior of a session and identify a possible cause of the problem. For example, you can use the Internal Events Tracing mode to check whether the session passes through the Service Broker components according to the Service Broker configuration.

When tracing internal events, Service Broker records the following information about the session into the log file:

- Timestamp that indicates when the session began
- Logger level
- Session ID
- Service Broker component which is currently handling the session
- Java class that logs the session
- Service Broker component and instance of the component that sends the session (source)
- Service Broker component and instance of the component that receives the session (destination)
- Message body

The following example shows the fragment of the log that contains the information about the session sent by the R-IM-OCF interworking module to the Orchestration Engine (OE):

Figure 15–1 Log of a Session Sent from R-IM-ASF to the OE



EVENT_ATTR_SRC_MODULE_INFO: Module : RIMOCF;Module ver : 2.0.0.0;Module instance: rimocf;Component : RIMOCF;Component ver : 1.0;

P-Asserted-Identity: <sip:calling@seagall.com>

specific_session_id: managed_14fb331a36b

Supported: 100rel

x-wcs-mobile-number: imsiNumber

To: <sip:called@convergin.com>

message_content_attr: [type=application/dcca+xml, body=[[object=AVPBodyCommand [m_commandAvps=[Auth-Application-Id=4][User-Name=0][Multiple-Services-Credit-Control=[Service-Identifier=5][Requested-Service-Unit=[CC-Time=5]]][Service-Information=[IMS-Information=[Calling-Party-Address=sip:calling@seagall.com][Called-Party-Address=sip:called@convergin.com][Node-Functionality=6][IMS-Charging-Identifier=icid-value]]][Event-Timestamp=856163706][Subscription-Id=[Subscription-Id-Type=1][Subscription-Id-Data=imsiNumber]][Service-Context-Id=32260@3gpp.org], m_commandName=CCR], type=application/dcca+xml, language=null]]]

x-wcs-module-name: rimocf

x-wcs-session-case: orig

message_method_attr: INVITE

x-wcs-timestamp: 856163706

message_top_route_attr: <sip:oe_instance.OE@wcs.convergin.com>

request_uri_attr: sip:called@convergin.com

CSeq: 1 INVITE

P-Charging-Vector: icid-value=icid-value

EVENT_ATTR_ID: managed_14fb331a36c

EVENT_ATTR_PROTOCOL_TYPE: SAL

sall-destination-attribute: SERVER

Route: [<sip:oe_instance.OE@wcs.convergin.com>][<sip:rimocf.RIMOCF@wcs.convergin.com>]

From: <sip:anonymous@ocsb-r-im-ocf.net>

EVENT_ATTR_INITIAL: true
]"

Full Session Tracing

You might need to use the Full Session Tracing mode when the Oracle support team requests you to log a session in order to investigate the issue.

When logging a session in the Full Session Tracing mode, Service Broker logs the following information about the session:

- All the information logged in the Internal Events Tracing mode.
- All debugging messages defined in the Service Broker programming code. These messages are for internal use by the Oracle support team.

Tracing a Session

To trace a session:

1. In the navigation tree, expand **OCSB**.
2. Expand the **Processing Tier** node.
3. Expand the **Tier Management** node.
4. Click the **Overload and Tracing** node.
5. Click the **Session Tracing** tab.
6. In the **User-id** field, enter the string that Service Broker should search in the initial request. See "About the Format of the Search String" for more information.
7. From the **Session Granularity** list, select one of the following:
   - When you want to trace a message, select **INTERNAL_EVENTS**. See "Internal Events Tracing" for more information.
   - When you want to trace the session, select **FULL_SESSION**. See "Full Session Tracing" for more information.
8. Click **Apply** to save your changes.

16

Remote Monitoring Service Broker with SNMP

This chapter describes how to use Simple Network Management Protocol (SNMP) to monitor Service Broker.

About Service Broker SNMP

You can monitor Service Broker remotely using the Simple Network Management Protocol (SNMP). The Service Broker supports SNMP Version 1 (SNMPv1), SNMP Version 2 (SNMPv2c), and SNMP Version 3 (SNMPv3).

SNMP management is based on the agent/manager model. The agent resides on the managed resource and provides information to one or more remote managers. In a Service Broker domain monitored by SNMP, an agent runs on each Signaling Server and Processing Server.

An SNMP agent provides information to managers by responding to queries or by sending unsolicited notifications (traps). SNMP queries can retrieve information on Service Broker activities, such as the number of SIP transactions processed and the length of time a module has been running.

The SNMP agent generates a trap when it detects certain predefined events or system conditions. For example, the Service Broker agent can send a trap when the server starts up or when an application error occurs. The agent sends traps to any SNMP manager that you specify as a trap destinations.

By default, the Service Broker SNMP agent sends notifications as SNMPv2c traps. It can also send traps in the format of SNMPv1 or SNMPv3 traps. Trap-forwarding groups enable you to send traps in different format, so that different trap destinations can receive traps in the version it supports.

The Service Broker domain allows you to individually configure SNMP settings for each Processing or Signaling Server in your deployment, as described in the following sections.

Service Broker SNMP messages are based on JMX notifications. Any JMX notification that is generated by the managed servers can be mapped to an SNMP trap. As long as the notification type is known it can be mapped.

About the Service Broker MIBs

A management information base (MIB) module defines the properties of the system that can be monitored by SNMP. When imported into MIB browsers or management systems, the MIB allows for automated discovery of the properties (or managed objects) that can be monitored.

You are required to create your own Service Broker and Axia MIBs. For backward compatibility, legacy MIBs are included in the Oracle Communications Service Broker Media Pack.

The Service Broker MIB modules are:

- **ocsb.mib**: Defines managed objects associated with Service Broker runtime and management components.
- **axia.mib**: Defines managed objects associated with the underlying Axia platform.

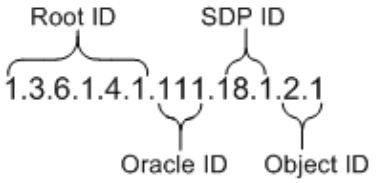
Note: Service Broker MIB objects are read-only. You cannot modify a Service Broker configuration using SNMP.

About Service Broker SNMP Object Identifiers

Each managed object defined in a MIB has a unique object identifier (OID). An OID consists of a series of dot-delimited numbers. Figure 16–1 shows the elements of an OID.

The example shows the four parts of the full object identifier. The parts first specify the Root ID which is 1.3.6.1.4.1. Next is the Oracle ID, which is 1.1.1. The SDP ID is 18.1. The final part is the Object identifier, which in this example is 2.1.

Figure 16–1 Elements of an OID



All OIDs for Service Broker objects have the same root ID (1.3.6.1.4.1) and Oracle enterprise ID (111). The remaining parts of the OID identify the product group (Service Delivery Platform (SDP) in this case) and the object ID, which may be qualified by an object group.

For example, the OID for the object group relating to statistics on SIP activity is:

sipNetworkChannelStatistics: 1.3.6.1.4.1.111.18.1.5

A managed object within the **sipNetworkChannelStatistics** group that provides statistics for TCP connections is:

sipNetworkChannelStatisticsTcpConnections: 1.3.6.1.4.1.111.18.1.5.4

You can use a MIB browser to view the structure and contents of the Service Broker MIB modules, including information on each managed object, such as its OID, syntax, and status.

Service Broker Managed Objects

The MIB modules that are shipped with Service Broker define both queriable objects and traps. Queriable objects provide extensive information on the activities of Service Broker and its components, including those of Interworking Modules, the Orchestration Engine, the Diameter adaptor, and management components.

Traps provide information on events associated with Service Broker. The Supported traps are:

- **serverRunningNotification**
- **serverStoppingNotification**
- **serverJoinNotification**
- **serverLeavingNotification**
- **bundleStartNotification**
- **bundleActiveNotification**
- **bundleStopNotification**
- **bundleErrorNotification**
- **diameterConnectionUpNotification**
- **diameterConnectionDownNotification**

Configuring SNMP with the Administration Console

You can use the Administration Console to create your own mappings from JMX notifications to SNMP traps. The Administration Console settings include the available JMX notifications, general SNMP settings for Managed Servers, and trap destinations.

The legacy, out-of-the-box SNMP traps do not populate the Administration Console GUI panels. You can optionally use them or re-write them in the Administration Console where they will be visible and editable.

Steps for Creating a JMX-based SNMP Trap

The essential steps for creating a JMX-based SNMP trap definition are as follows:

- Determine which runtime MBeans you want to use as the basis for your SNMP traps. See the *Oracle Communications Service Broker Configuration and Runtime MBean Java API Reference.*
- For the runtime MBeans you selected in the above step, configure the runtime MBean notification criteria for these indicators: counters, gauges, statuses. See "Understanding Notifications" for more information.
- Register the runtime MBeans to receive notifications. See "Registering for Notifications" for more information.
- Use the Administration Console to specify JMX notification types you want to map to SNMP traps and to configure other SNMP settings. See Table 16–7 for examples of JMX Notification types.
- Create a Service Broker MIB based on the SNMP traps you have configured. Obtain a MIB registration point from the IANA (Internet Assigned Numbers Authority).

Accessing SNMP Configuration Settings

To access the SNMP configuration settings in the Administration Console, follow these steps:

1. In the domain navigation tree, expand the **OCSB** node.
2. Expand **Domain Management**.

3. Select the **SNMP** node.

The SNMP configuration pane appears.

Configuring the SNMP Agent

The SNMP agent configuration determines the SNMP behavior for each Processing and Signaling Server in the managed domain.

By default, the domain defines an existing SNMP agent configuration in which the agent is disabled by default. Since it does not specify a target Managed Server, it applies to all servers in the cluster. To enable the SNMP agent on the servers, you enable the agent in the existing configuration or add your own, as described below.

To enable the default SNMP agent configuration:

1. In the SNMP node, click the **Agent** tab.
2. Select the existing agent configuration instance in the table.
3. In the **Enabled** field, enter **true**.
4. Modify other settings as desired. See Table 16–1 for information on the configuration settings.
5. Click **OK**.

To add SNMP agent definitions:

1. In the SNMP node, click the **Agent** tab if it does not already appear.
2. Click **New**.
3. In the New dialog, provide values for the fields listed in Table 16–1.

Table 16–1 SNMP Agent Configuration Settings

Fields	Description
Target	The name of the Managed Server to which the SNMP configuration applies. If you leave this field empty, the configuration applies to all Managed Servers in the domain.
Enabled	Boolean value that indicates whether the SNMP agent is active. Required. Possible values are: ■ **true**: Enables the SNMP agent. ■ **false**: Disables the SNMP agent. By default, the agent is disabled.
Port	The number of the port on which the Service Broker SNMP agent listens for queries from managers. Required. Conventionally, SNMP agents listen for requests on port 161. The default value is 8001.
Version	The SNMP version to use for the agent. Required. The default version is SNMPv2. Possible values are: ■ **V1**: Sets the SNMP version to SNMPv1. ■ **V2c**: Sets the SNMP version to SNMPv2c, or Community-Based Simple Network Management Protocol version 2. ■ **V3**: Sets the SNMP version to SNMPv3.

Table 16–1 (Cont.) SNMP Agent Configuration Settings

Fields	Description
Logging Level	The SNMP agent logging level as an integer. Required. Possible values are from 1 through 6, with the values corresponding to the following logging levels: ■ **1**: Only fatal events are logged. ■ **2**: Error-level events. ■ **3**: Warning-level events. ■ **4**: Informational-level events. ■ **5**: Debugging-level events. ■ **6**: Trace-level events.

4. Click **Save** to save your settings.

The new agent configuration appears in the table.

Configuring SNMP Access Control Restrictions

The access control table specifies access control restrictions that apply to queries to SNMP agents. The agent can authenticate incoming requests based on the community string provided in the request or by the IP address of the source. This access control mechanism applies to SNMPv1 and SNMPv2c.

To configure SNMP-related access control items, follow these steps:

1. In the SNMP configuration pane, click the **Access Control Table** tab.
2. From the **Parent** menu, choose the SNMP agent instance to which this configuration applies. The parent menu references an agent in the following form:

   SnmpConfig.configuration[*n*]

   Where *n* is the ID of the agent configuration as shown in the Agent Configuration pane.
3. Click the **New** button.
4. Provide values for the following fields.

Table 16–2 SNMP Access Control Settings

Fields	Description
aclCommunity	The community string required for query access. Required. In SNMP, the community string is used to establish trust between the agent and manager.
aclAccess	The authorization level for SNMP managers who match this community. Required. Possible values are ■ 0: No access. ■ 1: Read-only access. Because the Service Broker SNMP MIB defines all objects as read-only, option 2, read/write, is not supported.

Table 16–2 (Cont.) SNMP Access Control Settings

Fields	Description
aclManagers	The IP address or host name of managers who are allowed to access the agent. Required. Requests that provide the correct community string but originate from a source IP not specified in this parameter are blocked. Use 0.0.0.0 as the IP address to allow access to any manager who provides a matching community string, or provide a specific IP address. Use semi-colons (;) to separate multiple addresses.

5. Click **OK** to save your settings.

The new access control item appears in the table.

Configuring SNMPv1 and SNMPv2c Trap Destinations

The V1V2 trap forwarding table identifies SNMP manager trap destinations for traps in SNMPv1 and SNMPv2c format.

To configure SNMPv1 or SNMPv2c trap destinations, follow these steps:

1. In the SNMP configuration pane, click the **V1V2 Trap Forwarding Table** tab.
2. From the **Parent** menu, choose the SNMP agent instance to which this configuration applies. The parent menu references an agent in the following form:

   SnmpConfig.configuration[*n*]

   Where *n* is the ID of the agent configuration as shown in the Agent Configuration pane.
3. Click the **New** button.
4. Provide values for the following fields.

Table 16–3 V1V2 Trap Forwarding Table Settings

Fields	Description
Manager Host	The IP address or host name of the SNMP manager to which the agent sends SNMPv1 or SNMPv2c traps. Required.
Manager Port	The port number on which the SNMP manager listens for traps. Conventionally, managers listen for traps on port 162. Required.
Version	The SNMP protocol version to use for the traps. Required. Possible values are: ■ 1: Sets the SNMP version to SNMPv1. ■ 2: Sets the SNMP version to SNMPv2c, or Community-Based Simple Network Management Protocol version 2. ■ 3: Sets the SNMP version to SNMPv3.
Community	The community string for the trap destination manager. The community string is used to establish trust between the agent and manager.
Timeout	The notification transmission timeout period, in milliseconds. If the time expires, the agent considers the transmission to have failed and may re-attempt the transmission based on the **retries** value.

Table 16–3 (Cont.) V1V2 Trap Forwarding Table Settings

Fields	Description
Retries	The number of attempts that the agent makes to send a notification. If set to 0 or the maximum number or retries has been reached, the notification is not re-attempted and the notification is considered to have failed.

5. Click **OK** to save your settings.

The new trap forwarding settings appear in the table.

Configuring SNMPv3 Trap Destinations

The V3 trap forwarding table identifies SNMP manager trap destinations for traps in SNMPv3 format.

To configure SNMPv3 trap destinations:

1. In the SNMP configuration pane, click the **V3 Trap Forwarding Table** tab.
2. From the **Parent** menu, choose the SNMP agent instance to which this configuration applies. The parent menu references an agent in the following form:

   SnmpConfig.configuration[*n*]

   Where *n* is the ID of the agent configuration as shown in the Agent Configuration pane.
3. Click the **New** button.
4. Provide values for the following fields.

Table 16–4 V3 Trap Forwarding Table Settings

Fields	Description
Manager Host	The IP address or hostname of the SNMP manager to which the agent sends SNMPv3 traps. Required.
Manager Port	The port number on which the SNMP manager listens for traps. Conventionally, managers listen for traps on port 162. Required.
Version	The SNMP protocol version to use for the traps. Required. Possible values are: ■ 1: Sets the SNMP version to SNMPv1. ■ 2: Sets the SNMP version to SNMPv2c, or Community-Based Simple Network Management Protocol version 2. ■ 3: Sets the SNMP version to SNMPv3.
Community	The community string for the trap destination manager. The community string is used to establish trust between the agent and manager.
Username	The string specifying the user name of the manager user.
User Security Model	An integer value that specifies the manager's user security model. The only value supported by the Service Broker SNMP agent is 3, which specifies USM (User Security Model).

Table 16–4 (Cont.) V3 Trap Forwarding Table Settings

Fields	Description
Security Level	An integer that indicates which security features are applied to the message. You can require authentication and encryption of the trap content. Set the level using one of these options: ■ 1: Without authentication and without privacy (noAuthNoPriv). ■ 2: With authentication, but without privacy (authNoPriv). ■ 3: With authentication and with privacy (authPriv).
User Context Name	A string specifying the context of the manager user.
timeout	The notification transmission timeout period, in milliseconds. If the time expires, the agent considers the transmission to have failed and may re-attempt the transmission based on the **retries** value.
Retries	The number of attempts that the agent makes to send a notification. If set to 0 or the maximum number or retries has been reached, the notification is not re-attempted and the notification is considered to have failed.

5. Click **OK** to save your settings.

The new trap forwarding settings appear in the table.

Configuring JMX Notification Mappings

The SNMP mappings determine which JMX notifications are used as the basis for your SNMP traps.

The **General** tab is where you define general settings for the mappings. You need to fill in at least one parent setting definition before you can configure the trap mappings.

The **Trap Mappings** tab is used to map an available JMX notification to a KeyId, Name, and Trap sub OID that you can use in your MIB to define an SNMP trap.

To configure SNMP traps General settings, follow these steps:

1. In the SNMP configuration pane, click the **Mappings** tab.
2. Click the **New** button.
3. Provide values for the following fields.

Table 16–5 General SNMP Mappings Settings

Fields	Description
Name	Name to identify the mapping. This value has no functional meaning.
Enterprise OID	Corresponds to the OID that is assigned by IANA. Typically this is an OID that is already owned by the enterprise. Example: The Oracle Enterprise OID is 111.
Module OID	Corresponds to the OID of a module defined in the scope of the enterprise. The value can be a single number or multiple numbers separated by a dot. For example '45' or '18.1'.
Notifications OID	This is a number that defines the OID that is used for all notifications in the scope of the module. For example: '1' or '94'.

To configure SNMP Trap Mappings settings, follow these steps:

1. In the SNMP configuration pane, click the **Mappings** tab then the **Trap Mappings** tab.
2. Click the **New** button.
3. Provide values for the following fields.

Table 16–6 JMX Notifications Mapping Settings

Fields	Descriptions
Name	Name to identify the mapping. This value has no functional meaning.
Notification type	The notification type using this dotted string syntax: *abbreviated_mbean_name*.*attribute*.*cause* Where: *abbreviated_mbean_name* is the name of a runtime MBean with the **get** prefix and **mbean** suffix removed. *attribute* is an attribute of the runtime MBean. *cause* is the type of change that caused the notification from Table 14–6 and Table 14–7. See "Examples of JMX Notification Types" for examples, and "Notification Structure" for more information on the syntax.
Trap sub OID	The sub OID of the trap. The complete OID format for the trap is as follows (example): 1.3.6.1.4.1.[enterpriseOID].[moduleOID].[notificationsOID].[sub OID]

Examples of JMX Notification Types

Table 16–7 provides examples of some JMX Notification types. Any JMX notification generated by the managed servers can be mapped to an SNMP trap. As long as the notification type is known it can be mapped.

Table 16–7 JMX Notification Type Examples

Notification Type	Description
SsuRemotePointCodeRuntime.Status.StateEntered	This notification is sent when the SS7 Remote Pointcode Status Changed.
NetworkEntity.Status.StateEntered	This notification is sent when the SIP Network Entity Status Changed.
SystemCountersRuntime.InitialRequestCount.highThresholdCrossed	This notification is sent when the number of Initial Requests Crossed High Threshold Value.
SystemCountersRuntime.InitialRequestCount.clearedHighThreshold	This notification is sent when the number of Initial Requests Ceased High Threshold Value.
SystemGaugeRuntime.SessionGauge.highThresholdCrossed	This notification is sent when the number of Open Sessions Crossed High Threshold Value.
SystemGaugeRuntime.SessionGauge.clearedHighThreshold	This notification is sent when the number of Open Sessions Ceased High Threshold Value.

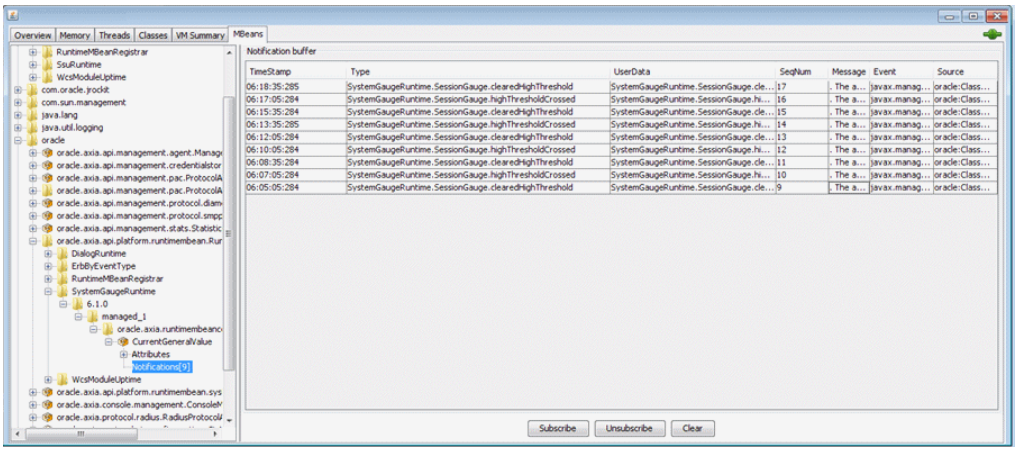
Table 16–7 (Cont.) JMX Notification Type Examples

Notification Type	Description
OeRuntime.UnsuccessfulApplicationTriggeringCount.highThresholdCrossed	This notification is sent when the number of Unsuccessful Application Triggering that the OE attempted Crossed High Threshold Value.
OeRuntime.UnsuccessfulApplicationTriggeringCount.clearedHighThreshold	This notification is sent when the number of Unsuccessful Application Triggering that the OE attempted Ceased High Threshold Value.
OprRuntime.UnsuccessfulQueryCount.highThresholdCrossed	This notification is sent when the number of Unsuccessful Queries that an OPR attempted to execute Crossed High Threshold Value
OprRuntime.UnsuccessfulQueryCount.clearedHighThreshold	This notification is sent when the number of Unsuccessful Queries that an OPR attempted to execute Ceased High Threshold Value

JMX Notifications Shown in JConsole

Figure 16–2 shows an example of a notifications buffer registered on an attribute of the runtime MBean SystemGaugeRuntime. The indicator monitored is a gauge that periodically crosses and ceases to cross a defined threshold.

Figure 16–2 JMX Notifications Buffer



17

Viewing Service Broker SDRs

This chapter explains how you can monitor Oracle Communications Service Broker using Service Data Records (SDRs). This chapter is applicable to the following products:

- Oracle Communications Service Controller
- Oracle Communications Online Mediation Controller

Important: Oracle Communications Service Policy Controller uses a different mechanism and format for handling SDRs. For more information, see the *Oracle Communications Policy Controller Implementation Guide*.

Understanding Service Data Records

An SDR is a set of parameter-value pairs that provide information on how service activation and delivery are performed through Service Broker.

An SDR is generated by the OE for each session. You will find SDRs on each Processing Server where an OE instance runs.

The OE stores SDRs in text files. The file name convention is as follows:

`ocsb.sdr.oe.`*`processing-server-host-name`*`.`*`SN`*

- *processing-server-host-name* is the name of the processing server where the OE runs
- *SN* is a file serial number, starting from 1

For example:

```
ocsb.sdr.oe.sb-processing01.17
```

Each file contains multiple SDRs. A file is closed when it reaches a preconfigured file size, at which time a new file is created. The default file size is 100 KB.

When the number of files reaches a preconfigured maximum, the oldest file is deleted with the addition of each new file. The default maximum number of files is 10.

If you need to store SDR files for longer periods of time, you should fetch the SDR files daily and move them to a separate system.

Configuring SDR Logging

Depending on your system capacity, you may want to modify maximum SDR file size and the maximum allowed number of SDR files. You can also disable writing SDRs to files if you do not need them.

SDR logging is controlled by Log4J. Service Broker includes preconfigured Log4J parameters with key-value pairs that define SDR logging.

The Log4J Configuration MBeans reflect the structure of the Log4J XML configuration file. See Log4J documentation at:

http://wiki.apache.org/logging-log4j/Log4jXmlFormat

The MBean Object Name is:

oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig

The default **configuration** MBean has an object name with **name1** set to **configuration[0]**:

oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig,name1=configuration[0]]

Setting the Maximum File Size and Number of Files

Using the Scripting Engine or an MBean browser:

1. Locate the Log4J Configuration MBean.
2. Locate the **configuration** MBean with object name:

   oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig,name1=configuration[0]]
3. Locate the **appender** MBean that has the attribute **name** set to **oe_file**.

   Example Object Name:

   oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig,name1=configuration[0],name2=appender[2]
4. Locate the **param** MBean with the attribute **name** set to **MaxFileSize**.

   Example Object Name:

   oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig,name1=configuration[0],name2=appender[2],name3=param[1]
5. Set the attribute **value** for the **param** MBean to the maximum file size. Default value is 100KB.
6. Locate the **param** MBean with the attribute **name** set to **MaxBackupIndex**.

   Example Object Name:

   oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig,name1=configuration[0],name2=appender[2],name3=param[2]]
7. Set the attribute **value** for the **param** MBean to the number of files. Default value is 10.

Disabling Logging of SDRs

To disable writing SDRs to files, change the log level of the SDR logger from **trace** to **info** or any higher log level.

Using the Scripting Engine or an MBean browser:

1. Locate the Log4J Configuration MBean.
2. Locate the **configuration** MBean with object name:

   oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig,name1=configuration[0]]
3. Locate the **logger** MBean that has the attribute **name** set to **com.convergin.oe.common.datarecord.OeSpooler**.

   Example Object Name:

   oracle:type=oracle.axia.cm.ConfigurationMBean,name=oracle.axia.logging.log4jconfig,version=1.0.0.0,name0=log4jConfig,name1=configuration[0],name2=logger[5]
4. Locate the **level** MBean for the **logger** MBean.
5. Set the attribute **value** for the **level** MBean to **trace** or **info**. Default value is **trace**, which enables SDR logging.

Service Data Record Format

Each SDR consists of parameter-value pairs that store information about service activation and delivery performed through Service Broker. Table 17–1 describes the parameters available in each SDR.

Table 17–1 Service Broker SDR Fields

Field	Description
ModuleType	Specifies the type of Service Broker module that generated the SDR. The value of this field is always OE.
ModuleVersion	Specifies the OE version
RecordType	Specifies the type of session that triggered the OE: Possible values: ■ CallControl: The OE was triggered by a call ■ SmsControl: The OE was triggered by a message (for example, SMS)
ModuleInstanceName	Specifies the module instance name.
Server	Specifies the name of the Processing Server where the OE runs.
CallDirection	Specifies the session direction. Possible values: ■ Incoming ■ Outgoing
CallReferenceNumber	Specifies the unique session reference number.
ChargingVector	Specifies the call p-charging-vector, as defined in IETF RFC 3455.
SBCorrelationID	Specifies a unique identifier generated by the first Service Broker module in the session path.
SubscriberID	Specifies the identifier of the subscriber for whom the service is triggered.

Table 17–1 (Cont.) Service Broker SDR Fields

Field	Description
ServiceKey	Specifies the service identifier. This field is displayed only when the OE is triggered through the IM-SCF CAP or IM-SCF CS1.
CallingPartyNumber	Specifies the calling party number.
AdditionalCallingPartyNumber	Specifies the calling party number. Usually this number is identical to the value of the CallingPartyNumber field.
CalledPartyNumber	Specifies the called party number.
OPR	Specifies the Orchestration Profile Receiver that the OE used to obtain the orchestration profile. For more information about the types of OPRs, see About Orchestration Profile Receivers in *Oracle Communications Service Broker Concepts Guide*.
OLID	Specifies the unique identifier of the orchestration profile that the OPR downloaded. When the LSS is used, this field shows the ID field given to an LSS profile. When the orchestration logic is not found, this field is set to Not Found.
OLP	Specifies the OLP that was used.
ServiceStartTime	Specifies the time when the session started (that is when the OE was triggered)
ServiceTermTime	Specifies the time when the Service Broker finished handling the session
Application	Specifies the application that the Service Broker triggered in the following format: *Application Name*; *Time*; *Application Response*, ■ *Application Name* specifies the application that the OE triggered ■ *Time* specifies the time when the OE triggered the application ■ *Application Response* specifies the SAL message returned from the application. Possible values: INVITE, 302, 4xx, or 5xx. For example: `SBX@domain.com; 2009-02-13T07:29:28.482-0600; 302`
ImInfo	Information received from IMs in the session path inside Service Broker. The information is dynamic and varies, depending on the IM that provided the information. Field format: *im-type-*; *info-tag-value-string* For example: ImInfo: IMOCF; sid=35263; requm=3; nonChargeDuration=25 The values in the info-tag-value-string are listed in IM-Generated Tags.

The following is an example of an SDR:

```
ModuleType : oe
ModuleVersion : 1.0
RecordType : CallControl
ModuleInstanceName : oe_instance
Server : sb_processing01
```

```
CallDirection : Outgoing
CallReferenceNumber : 36011211571409664
ChargingVector : icid-value=360112115714096647FF001; icid-generated-at=sb.ora.com
SBCorrelationID : null
SubscriberID : <sip:2425540022@wcs.convergin.com;noa=national>
ServiceKey : 11
CallingPartyNumber : <sip:2425540022@wcs.convergin.com;noa=national>
AdditionalCallingPartyNumber :
"sipp"<sip:sipp@192.168.0.170:5060>;tag=1260965860664
CalledPartyNumber : "sut"<sip:service@192.168.0.170:2345>;tag=1260965860665
OPR : LSS
OLID : 101
OLP : ifc
ServiceStartTime : Wed Dec 16 14:17:41 IST 2009
ServiceTermTime : Wed Dec 16 14:17:52 IST 2009
Application : <sip:imasf_instance.IMASF@convergin.com;lr> ; Wed Dec 16 14:17:42
IST 2009 ; INVITE
```

IM-Generated Tags

Service Broker IMs generate session-related data describing IM activity during the session. The OE receives this data and incorporates the data into the SDR. The data consists of a range of tags, which vary according to the IM type.

Table 17–2 describes the tags generated by the IM-OCF.

Table 17–2 IM-Generated Tags

IM Type	Tags
IMOCF	■ noChargeDuration=X The time that elapsed from the last successful CCR request to the end of the session. This time can be used at a later time for provisioning in the OCF (if it fails), for offline charging. ■ DiameterSessionID=Y The ID of the degraded mode session. ■ FailedCCRNumber=Z The number of the CCRs which failed to provide a response during a session.

18

Implementing Overload Protection

This chapter explains how to protect Oracle Communications Service Broker from overload.

About Overload Protection

In some cases, such as unanticipated traffic peaks or failure of a network hardware or software component, the load on Service Broker modules can increase significantly. This can cause a situation known as system overload in which Service Broker modules have insufficient resources to handle new sessions. If overload is not handled correctly, the system can fail and lose critical data.

To handle increased amounts of traffic without damaging operations of the entire system, Service Broker provides an overload protection mechanism. This mechanism operates in Processing Domains where you can define criteria for overload detection.

By default, an overload condition is triggered by too many active sessions or initial requests. The system rejects initial requests while the overload lasts. In addition to rejecting initial requests, Service Broker provides the capability to customize how the system behaves if overload occurs.

Using Gauges and Counters as Key Overload Indicators

When you configure gauges and counters as key overload indicators, Service Broker triggers overload protection if threshold values are crossed as measured by those indicators. You can select any of the counters and gauges provided by Service Broker to serve as key overload indicators.

> **Note:** Consult with Oracle Technical Support if you have questions about which runtime MBeans are best suited to implement overload protection in your network environment.

When you configure overload protection, your settings are applied uniformly across all managed servers in the domain. Usually, server load balancing allocates traffic "fairly" across servers. However, it is possible for a single managed server in a domain to enter an overload condition while the other servers are functioning normally.

About System and Module Levels of Overload Protection

Service Broker overload protection can be configured at the system and module levels:

- System counters and gauges: These two indicators can detect and trigger an overload condition that might occur across any number of clustered managed servers or when deploying any combination of Service Broker products.
  - **`SystemCountersRuntimeMBean.SessionGauge`**
  - **`SystemCountersRuntimeMBean.InitialRequestCount`**

  The SessionGauge gauge represents the total number of active sessions on a single managed server (JVM). Sessions are application specific, for example there are separate Online Mediation Controller, Service Controller, and Policy Controller sessions.

  The InitialRequestCount counter represents the session creation rate on a single managed server (JVM). For example, the number of new sessions created per second on a single managed server.

  You configure the SessionGauge and InitialRequestCount indicators for the managed servers in the Administration Console. All managed servers share those configuration settings.

  There is no global overload protection status. However, if any managed server goes into an overload state, as defined by the shared configuration, the system stops accepting new sessions until the overload condition ceases.
- Module-level counters and gauges: These indicators are for specific modules. Using the Administration Console, you can configure these indicators for overload protection under the monitoring tabs which you access by expanding Platform, then OCSB, then Processing Tier, and then Interworking and Supplementary Modules.

Understanding the Essential Steps for Configuring Overload Protection

All of these steps must be followed to configure overload protection.

1. By default, the following gauge and counter are defined as your key overload indicators:
   - **`SystemCountersRuntimeMBean.SessionGauge`**: A gauge that measures the number of active sessions handled by a single managed server. The number of active sessions includes (if installed): Service Controller sessions, Online Mediation Controller sessions, and Policy Controller sessions.
   - **`SystemCountersRuntimeMBean.InitialRequestCount`**: A counter that measures the rate at which a managed server receives new sessions.

     The SessionGauge and InitialRequestCount measurements are per managed server but all of the managed servers share a common configuration. If any managed server goes into an overload condition, the system stops accepting new sessions until the overload condition ceases.
2. Identify Module-level counters and gauges you want to use as key overload indicators.

   Module-level counters and gauges are discussed in the *Service Controller Implementation Guide, Online Mediation Controller Implementation Guide,* and the *Policy Controller Implementation Guide*.
3. Configure Threshold Crossed Notifications details for your module-level counters and gauges.

   For each counter and gauge you want to use as a key overload indicator, define an upper threshold and ceased value threshold. For example, if the upper threshold

value is 100 and the ceased value is 90 then if 100 is crossed the system remains overloaded until the value goes below 90.

Specify a threshold name for each module-level counter or gauge. If you use either sessionGauge or initialRequestCount as the threshold name value you do not have to add these indicators to the Key Overload Indicators pane.

See Chapter 14, "Monitoring Service Broker Using Runtime MBeans" for more information about setting thresholds.

4. Configure Key Overload Indicators.

   After you have configured the module-level counters and gauges you want to use for overload protection you need to specify that they are to be used by Service Broker as Key Overload Indicators.

   You do this by using the Administration Console. Expand Tier Management, and then use the Key Overload pane to define your key overload indicators.

   Important: Service Broker activates overload protection when any of your key overload indicator crosses its upper threshold.

5. Customize overload protection behavior.

   The built-in behavior of Service Broker is that if an overload condition occurs, the system continues to handle all active sessions but rejects initial requests until the overload condition ceases.

   In addition to the default protection behavior, you can customize how Service Broker responds to SIP and Diameter network entities that attempt to establish sessions during a system overload.

   Example: You can define the type of error and value of the SIP Retry-After header field that Service Broker uses to respond to newly established SIP sessions.

   You can customize overload protection behavior by using the Administration Console. Expand Tier Management, then Overload and Tracing, and then customize overload protection in the Overload Protection Methods pane.

Configuring Key Overload Indicators

The following sections describe in detail how to configure Key Overload Indicators.

Configuring Threshold Crossed Notifications Rules

This section describes how to create Threshold Crossed Notifications rules for overload protection. The components of these rules specify MBean type, threshold name, crossed and ceased threshold values, and other fields.

The following steps are applicable to both systemwide and module-level counters and gauges. There are only two systemwide overload indicators: sessionGauge and InitialRequestCount. The default settings for these indicators should usually not be changed.

To transform the counter or gauge you configure in this section to be a key overload indicator you must match the threshold name value you set under the Monitoring tab with the threshold name value in the Key Overload indicators pane.

For example, the default key overload indicator threshold name **`sessionGauge`** matches the threshold name value in the default threshold crossed notifications rule also sessionGauge.

To configure Threshold Crossed Notification Rules do the following:

1. In the navigation tree, expand the **OCSB** node.
2. Expand **Processing Tier**.
3. Do either of the following:
   - To configure System-level Counters and Gauges: Expand Tier Management, then Monitoring, and then Monitoring. **Note**: You cannot add more counters and gauges in addition to the default sessionGauge and InitialRequestCount indicators. However, if required you can modify details such as crossed and ceased threshold values.
   - To configure Module-level Counters and Gauges: Expand Interworking or Supplementary Modules, and then expand the module for which you want to configure a counter or gauge as an overload indicator.
4. In the **Monitoring** tab, select **Threshold Crossed Notifications**.
5. Be sure you have selected **Lock & Edit** and then click **New**.
6. In the **Threshold Name** field, enter a string that names the threshold. This value is referenced by the key overload indicators.
7. For the **Enable threshold** field, select **True** or **False**. Only enabled thresholds are considered for overload protection.
8. In the **MBean Type** field, enter the type of MBean. For system gauges use SystemGaugeRuntime and for system counters use SystemCountRuntime.
9. For the **Counting Type** field, select the Counting method. For gauges use CurrentGeneralValue and for counters use CurrentIntervalDeltaValue.
10. In the **MBean Attribute** field, enter an MBean attribute. For SystemGaugeRuntime use SessionGauge and for SystemCountRuntime use InitialRequestCount.
11. In the **Threshold class** field, enter **High**. Crossing a low threshold does not cause an overload state.
12. In the **Threshold Value** field, enter an integer value which when crossed triggers an overload state.
13. In the **Threshold ceased value** field, enter an integer value which when crossed the triggered threshold ceases. This value is applicable only to gauges.
14. In the **Threshold crossed message** field, enter a message included in the threshold notification.
15. In the **Threshold ceased message** field, enter a message included in the threshold ceased notification.
16. In the **Server filter** field, leave it empty or use a regular expression to filter on a managed server. For example "managed_1" or "server."
17. In the **Resource filter** field, enter a name for the indicator.
18. Click **Apply**.

Specifying Your Key Overload Indicators

Identify the module-level counters and gauges you want to use for overload protection. Use the Administration Console to list the names and threshold names for these indicators.

The Overload Protection pane is pre-populated with these two systemwide indicators:

- **SystemCountersRuntimeMBean.SessionGauge**
- **SystemCountersRuntimeMBean.InitialRequestCount**

To specify module-level Key Overload Indicators do the following:

1. In the navigation tree, expand the **OCSB** node.
2. Expand **Processing Tier**.
3. Expand **Tier Management**.
4. Select **Overload Protection**. The **Key Overload Indicators** pane appears.
5. Be sure you have selected **Lock & Edit** and then click **New**.
6. In the **Name** field, enter a name for the indicator.
7. In the **Threshold Name** field, enter a string that references the threshold.

   Multiple indicators at the system or module levels can use the same Threshold Name. In this situation, all matching crossed thresholds will be considered to indicate an overload state.

   Example: If any module-level counter or gauge uses either sessionGauge or initialRequestCount as the threshold name value, you do not have to add these module-level indicators to the Key Overload Indicators pane.

   However, the module-level settings (e.g. crossed threshold value) will override the platform-level settings for that individual module only.
8. Click **Apply**.

Configuring General Monitoring Parameters

This section describes how to configure general attributes for overload protection notifications.

These attributes can be configured both at the system and module levels by doing the following: Expand Tier Management, then Monitoring, and then select the General tab. For configuring IMs, expand Interworking Modules, then the IM you want to configure, then Monitoring, and then select the General tab.

Table 18–1 describes the configuration parameters on the Monitoring General tab. At the Tier level these parameters affect only SessionGauge and InitialRequestCount. At the module level the parameters affect all runtime MBeans in the module.

Overload protection is disabled by default.

Table 18–1 General Overload Configuration Parameters

Name	Description
Enable runtime MBeans	Disables the platform-level runtime MBeans so you can neither poll them for values or get notifications.
Enable Notifications	Disables only notifications, so you can still poll values from the MBean.
Counter Interval (sec)	This parameter specifies the length of the interval in seconds. Note: This parameter is not configurable at the module level.

Table 18–1 (Cont.) General Overload Configuration Parameters

Name	Description
Notification trigger interval (sec)	Sampling interval in seconds for checking notifications. For example, if the Counter Interval is set to 10 seconds and the Notification trigger interval is set to 2 seconds for each counter interval the system will determine 5 times whether the threshold value has been crossed.

Configuring the Overload Protection Methods

When system overload occurs, Service Broker rejects new sessions and sends response messages to the network entities that attempted to establish the new sessions.

In the Overload Protection Methods pane you can configure how Service Broker responds to attempts by SIP and Diameter network entities to establish new sessions.

Table 18–2 describes configuration parameters on the Overload Protection Methods subtab.

Table 18–2 Overload Protection Methods

Name	Type	Description
Enabled	BOOL	Specifies whether the overload protection methods specified in this table are enabled. Possible values: ■ TRUE ■ FALSE
SIP Response Status Code	STRING	Specifies a SIP error that Service Broker returns to a SIP network entity when Service Broker declines an attempt to establish a session. Default value: 503
SIP Retry-After	STRING	Specifies the value that Service Broker sets in the Retry-After header of the error response sent to the network entity. This value defines how long the network entity waits before it retries to establish a session. Default value: 300
Diameter Response Result Code	STRING	Specifies a response Result Code AVP that Service Broker returns to a Diameter network entity when Service Broker declines the attempt to establish a session. Default value: 5012
Web Service Response Status Code	INT	Specifies an error code that Service Broker returns to a Web service network entity when Service Broker declines the attempt to establish a session. Default value: 503
SAL Response Status Code	INT	Specifies an error code that Service Broker returns to a SAL application when Service Broker declines the attempt to establish a session. Default value: 503

Best Practices

In a domain that deploys more than a single product, for example Service Controller, Online Mediation Controller, and Policy Controller, all products will impact the **`InitialRequestCount`** counter and the **`SessionGauge`** gauge provided by the platform.

Accordingly, you should take this situation into account when configuring the thresholds for overload protection.

A

System Administrator's Reference

This appendix contains reference information on directory structures and directory contents, along with details about the installer files, start-scripts, and JDKs.

Details for Administration Server

This section specifies the authentication methods, directory structure, directory contents, and start-scripts for the Administration Server.

Authentication Methods

The Administration Server enables different authentication methods for these clients:

- Administration Console
- Remote JMX-client
- Scripting Engine

Administration Console

The Administration Console supports a single user. By default the security for this user includes Digest Authentication and an SSL connection between the Administration Console and the Administration Server.

The first time you start the Administration Server you are prompted to supply a user name and password. For example:

```
# ./admin.sh /<Domain Path>
Please enter username and password that will be required to access the web
interface.
Enter Username: User
Enter Password: *******
```

These login credentials must be reentered for each Administration Console session.

To reset the user name or password, you must restart the Administration Server.

An alternative method of authentication is available by using a credential store. After product installation, but before starting the Administration Server you can configure this type of security. See the Chapter "Administering Credential Stores" in the *Oracle Communications Service Broker Security Guide*.

Remote JMX-Client

A remote JMX-client, such as JConsole, provides various options for securing its connection to the server. You use standard Java properties for remote access enabling SSL, users, passwords, and roles.

The Java documentation is located here:

http://docs.oracle.com/javase/6/docs/technotes/guides/management/agent.html

System properties that are described you can set using AXIA_OPTS environment variable.

Example A–1 Using No Security - Not Recommended

```
-Dcom.sun.management.jmxremote.port=1234 -Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
```

Note: The next example requires that you create the keystore, password, and access files. The location of the keystore/truststore is configured in properties/common.properties

Example A–2 Using SSL, Users, Passwords, and Roles

```
-Dcom.sun.management.jmxremote.port=1234 -Dcom.sun.management.jmxremote.ssl=true
-Dcom.sun.management.jmxremote.password.file=jmxremote.password
-Dcom.sun.management.jmxremote. access.file=jmxremote.access
```

Scripting Engine

You run scripts locally on the computer where you have direct access to the domain directory structure. The scripting engine is protected by your login authentication and permissions for the domain directory structure.

Directory Structure and Contents for the Administration Server

All administration server directories and contents are installed under the directory:

Linux and Solaris: *Oracle_home*/**ocsb61/admin_server**

Oracle_home is the Oracle home directory you defined when you installed the product.

Table A–1 describes the directory structure and the contents of the directory structure.

Table A–1 Directory Contents and Structure for Administration Clients Relative to Oracle_home/ocsb61

Directory	Description
admin_server	Top-level directory for all administration server clients. Contains start-scripts for: ■ Administration Server ■ Scripting Engine ■ Domain Web server ■ Database configuration Also contains files related to log4j: ■ **console.log** file is the default log file used for the administration clients. ■ **log4j.xml** defines logging properties used for the administration clients.
admin_server/applications	Created during start-up. Empty directory.
admin_server/extensions	Extensions to the Administration Server specific to the features installed.
admin_server/extensions_cef	Extensions to the Administration Server specific to the features installed.
admin_server/domain_ configuration	Contains these directories: ■ **/meta** - Contains metadata .xml files that support domain creation. ■ **/domain_configuration** - Supporting files for domain creation.
admin_server/modules	Contains all OSGi bundles for the administration clients, the Processing Server and the Signaling Server.
admin_server/osgi	Contains OSGi-specific configuration for the Administration Server processes.
admin_server/properties	It contains property files used by the start-scripts for: ■ The SVC and VPN applications. ■ Administration Server ■ Scripting Engine ■ Domain Web server ■ Hosted domains
admin_server/scripts	Contains these scripts used for domain creation: ■ create_domain_bundles.xml ■ create_domain.xml ■ create_hosted_domain.xml ■ define_ocsb_avps.xml ■ define_ocsb_loggers.xml ■ list_bundles.xml Contains the **/database** directory containing scripts for configuring databases for the Service Broker features that require it.
admin_server/utils	Contains utilities used by the SVC and VPN features.
admin_server/workspace	Contains metadata for administration clients.

Start Scripts

Table A–2 provides information about start-scripts for Service Broker.

Table A–2 Start-scripts for the Administration Clients

Script	Description
script.sh	Starts the Scripting Engine. **script.sh** calls **common.sh**. See "Using the Scripting Engine to Configure a Domain" for details.
start.sh	Starts the managed server.
web.sh	Starts the Administration Server. **web.sh** calls **common.sh**.
host.sh	Starts the Domain Web server. **host.sh** calls **common.sh**. See "Starting and Stopping Processing and Signaling Servers" for information on how to use the script.
common.sh	Starts the Administration Server, Scripting Engine, and the Domain Web server based on the environment variables set by the script that calls it. Defines the environment variables that send additional arguments to the JVM: ■ **AXIA_OPTS** - See "Using Wildcard Characters in Scripts" for more information. ■ **AXIA_MEM_OPTS** - Used to change the Java memory settings in the start script. The syntax for **AXIA_MEM_OPTS** is: **export AXIA_MEM_OPTS=-"-***memory_variable new_ memory_value***" ...** Where: *memory_variable* is the Java memory setting to change. *new_memory_value* is the amount of memory to reserve. For example, his command sets the Java minimum and maximum heap size settings to 1Gb: **export AXIA_MEM_OPTS="-Xms1024m -Xmx1024m"** For details on available JVM arguments refer to HotSpot and JRockit documentation: ■ http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html ■ http://docs.oracle.com/cd/E15289_01/doc.40/e15062/toc.htm

Property Files for the Administration Clients

Table A–3 lists property files in

*Oracle_home***/admin_server/properties** and their settings.

Table A–3 Property files used by the Administration Clients

Property File	Description
common.properties	Defines properties common to the: ■ Administration Server ■ Scripting Engine ■ Domain Web server The properties specified are: ■ axia.production.mode ■ axia.console.log4j.server.port ■ axia.ssl.cipher_suites ■ axia.admin.verify.hostname ■ https.cipherSuites ■ javax.net.ssl.keyStore ■ javax.net.ssl.trustStore ■ log4j.configuration ■ axia.console.password.validation.enabled ■ axia.console.password.validation.min_length ■ axia.console.password.validation.require_lower ■ axia.console.password.validation.require_upper ■ axia.console.password.validation.require_digit See the **common.properties** file and Table A–8 for details on the property settings. See Table A–9 for information on the security entries.
create_db_table.properties	Defines properties for the SVC and VPN applications. The properties specified are: ■ profile.db.server ■ profile.db.port ■ profile.db.dbname ■ profile.db.user
hosting.properties	Defines properties for the Domain Web server. The properties specified are: ■ axia.platform ■ org.eclipse.equinox.http.jetty.http.enabled ■ org.eclipse.equinox.http.jetty.http.port ■ org.eclipse.equinox.http.jetty.https.enabled ■ org.eclipse.equinox.http.jetty.https.port ■ org.eclipse.equinox.http.jetty.ssl.needclientauth ■ org.eclipse.equinox.http.jetty.ssl.keystore See **hosting.properties** and Table A–8 for details on the property settings. See Table A–9 for information on the security entries.

Table A–3 (Cont.) Property files used by the Administration Clients

Property File	Description
script.properties	Defines properties for the Domain Web server. The properties specified are: ■ axia.platform ■ org.eclipse.equinox.http.jetty.http.enabled ■ org.eclipse.equinox.http.jetty.http.port See **script.properties** file and Table A–8 for details on the property settings. See Table A–9 for information on the security entries.
admin.properties	Defines properties for the Administration Server. The properties specified are: ■ axia.platform ■ axia.require.domain ■ axia.digest.auth ■ org.eclipse.equinox.http.jetty.http.enabled ■ org.eclipse.equinox.http.jetty.http.port ■ org.eclipse.equinox.http.jetty.https.enabled ■ org.eclipse.equinox.http.jetty.https.port ■ org.eclipse.equinox.http.jetty.ssl.keystore ■ org.eclipse.equinox.http.jetty.other.info ■ org.eclipse.equinox.http.jetty.customizer.class See the **admin.properties** file and Table A–8 for details on the property settings. See Table A–9 for information on the security entries.

Details for Processing Servers and Signaling Servers

This section specifies the directory structure, directory contents and start-scripts for Processing Servers and Signaling Servers.

Directory Contents and Structure for Processing Servers and a Signaling Servers

Processing Servers and a Signaling Servers are installed under the directory:

*Oracle_home***/ocsb61/managed_server**

Oracle_home is the Oracle home directory you defined when you installed the product.

Table A–4 describes the directory structure and the contents of the directory structure.

Table A–4 Directory Contents and Structure for Processing Servers and Signaling Servers relative to Oracle_home/ocsb61

Directory	Description
managed_server	Top-level directory for a Processing Server and a Signaling Server. Contains start-scripts for the Processing Server and the Signaling Server. Contains the property file **server.properties**. Also contains files related to log4j: ■ **server.log** is the default log file used for the servers. ■ **log4j.xml** defines logging properties used for the servers. These files are relevant up the point in the platform life cycle when the bundle for the log4j service is started. After this point, this configuration is overridden by the configuration in the log4j service itself.
managed_server/config	Contains configuration data.
managed_server/modules	Contains all necessary bundles to start the OSGi framework and bundles for: ■ Platform logging service ■ log4j ■ Provisioning service The bundles in this directory are the minimal set necessary to initiate the server and load the contents of the domain configuration directory.
managed_server/osgi	A working directory for the Managed Server process.
managed_server/ss7	Contains binaries for the SS7 stacks for TDM and Sigtran.

Properties File for Managed Servers

Table A–5 gives information property files in:

Oracle_home/**managed_server/properties**

Table A–5 Property Files Used by Processing Servers and Signaling Servers

Property File	Description
server.properties	Defines properties common for Processing Servers and Signaling Servers. The properties specified are: ■ axia.platform ■ log4j.configuration ■ javax.net.ssl.keyStore ■ javax.net.ssl.trustStore ■ axia.admin.verify.hostname See the **server.properties** file and Table A–8 for details on the property settings.

Details for Domains

This section specifies the directory structure and directory contents for domains.

Directory Contents and Structure for Domains

Domain directories are created, one for each domain, under the domains home directory, by the domain creation script.

Domains_home/*Domain_dir*

Domains_home is the directory where you store all domain directories, also know as domain configuration directories. For example: /home/oracle/domains/

Domain_dir is where the domain configuration is stored

Domain directories are defined in the *domain_path* parameter when you run the domain creation script. Normally, all domain directories are created under the same Domains Home directory.

Table A–6 describes the directory structure and contents of the directory structure.

Table A–6 Directory Contents and Structure for Domains relative to Domains_home

Directory	Description
Domain_dir	Top-lever directory for a domain. Contains the domain configuration file **initial.zip**. Contains the properties file **domain.properties** (for Oracle internal use only). This directory is passed to the server start script and this is where a server takes its configuration from.
Domain_dir/modules	Contains all necessary bundles to start the domain functions: processing tier modules, signaling tier modules, or both.
Domain_dir/protected	Contains the domain credential file and the master passwords file protecting the credential file. Both files are encrypted.
Domain_dir/workspace	Contains domain configuration while it is being edited either through the Administration Console or configuration MBeans.

Environment Variables

Table A–7 gives information about the environment variables used.

Table A–7 Environment variables

Variable	Description
AXIA_OPTS	Defines any additional Java options to use. Can be used to create a domain in silent mode by setting values for AXIA_OPTS as command-line arguments.
AXIA_MEM_OPTS	Overrides the default memory settings for the JVM, such as heap size.

System Properties

Table A–8 describes the general system properties defined for Oracle Communications Service Broker. The security-related property entries are listed in Table A–9.

Table A–8 Description of System Properties

System Property	Description
axia.console.log4j.server.port	The port to use for static log4j XML logging service traffic. Set in **common.properties**
axia.platform	Defines the start mode. These default settings must not be changed: ■ **server** in **server.properties** ■ **web** in **admin.properties** ■ **script** in **script.properties** ■ **hosting** in **hosting.properties**
diameter.watchdog.for.dynamic.peers	Boolean. Defines whether the Diameter SSU should send Device-Watchdog-Request (DWR) commands to dynamic Diameter peers. **true** - Directs Diameter SSU to send DWR commands to dynamic peers. **false** - Stops Diameter SSU from sending DWR commands. This is the default setting. Use **AXIA_OPTS** to change this setting before starting the Signaling Servers server. This example sets this setting to **true**: **export AXIA_ OPTS="-Ddiameter.watchdog.for.dynamic.peers=true"** The Diameter SSU applies this property only when dynamic peers are allowed.
diameter.tcp.keepalive.for.client.peers	Boolean. Defines whether the TCP socket option **SO_ KEEPALIVE** for Diameter dynamic peers is enabled. **true** - Enables SO_KEEPALIVE. **false** - Disables SO_KEEPALIVE. This is the default setting. Use **AXIA_OPTS** to change this setting before starting the Signaling Servers server. This example sets this setting to **true**: **export AXIA_ OPTS="-Ddiameter.tcp.keepalive.for.client.peers=true"** The Diameter SSU applies this property only when dynamic peers are allowed.
log4j.configuration	The name of the static log4j XML configuration file. Set in **common.properties** for the administration tools. Set in **server.properties** for the Processing Server and the Signaling Server.

Table A–8 (Cont.) Description of System Properties

System Property	Description
org.eclipse.equinox.http.jetty.http.port	Specifies the HTTP port number the Jetty listens for HTTP traffic on if **org.eclipse.equinox.http.jetty.http.enabled** is set to **true**. Default value is **9000**. Set in: ■ **admin.properties** ■ **hosting.properties** The setting in **admin.properties** defines the port for the Administration Server. The setting in **hosting.properties** defines the port for the Domain Web server. This setting must correspond to the port defined when the domain configuration was created.
org.eclipse.equinox.http.jetty.http.enabled	Boolean. Specifies whether HTTP is used by the Jetty server. Set this property to: ■ **true** to use HTTP. ■ **false** to not use HTTP. Set in: ■ **hosting.properties** ■ **script.properties** ■ **admin.properties** Must always be set to **false** in **script.properties** and **admin.properties**.
org.eclipse.equinox.http.jetty.https.enabled	Boolean. Specifies if HTTPS is used by the Jetty server. Set this property to: ■ **true** to use HTTPS. ■ **false** to not use HTTPS. Set in: ■ **script.properties** ■ **admin.properties** Must always be set to **false** in **script.properties** and **admin.properties**.
org.eclipse.equinox.http.jetty.https.port	Specifies the HTTP port number to use for HTTP communication if **org.eclipse.equinox.http.jetty.https.enabled** is set to **true**. The default value is **9000**. Set in **admin.properties** and **hosting.properties**.
org.eclipse.equinox.http.jetty.other.info	Specifies which help-system to use for the Administration Console. Ignored, for future use. Set in **admin.properties**.
profile.db.dbname	Specifies the name of the profile database server used by the SVC and VPN features. The default value is **orcl**. Set in the **create_db_table.properties** file.

Table A–8 (Cont.) Description of System Properties

System Property	Description
profile.db.port	Specifies the port of the profile database server used by the SVC and VPN features. The default value is **1521**. Set in the **create_db_table.properties** file.
profile.db.server	Specifies the IP address of the profile database server used by the SVC and VPN features. There is no default value. Set in the **create_db_table.properties** file.
profile.db.user	Specifies the database user used by the profile database server. Used by the SVC and VPN features. The default value is **ocsb**. Set in the **create_db_table.properties** file.

Table A–9 lists the security-related property file entries. See Table A–8 for the other system property file entries.

Table A–9 System Security Properties

System Security Property	Description
axia.admin.verify.hostname	Boolean. Determines whether hostname verification is required for each administrator certificate connection. Default value is **true**. Set in **common.properties**.
axia.digest.auth	Boolean. Specifies whether to use *digest authentication* which is a standard defined by an IEEE RFC, **http://www.ietf.org/rfc/rfc2617.txt**. You have these options for setting the type of authentication used between the Administration Console and the Administration Server: ■ Leave **axia.digest.auth** on its default setting of true, which requires a username and password to authenticate the client. The username is sent in clear text, but the the password is encrypted using a nonce value. This is the most secure setting and is recommended for production deployments. ■ Change **axia.digest.auth=true** to **axia.basic.auth=true**. This requires a username and password to authenticate the client, but sends them unencrypted. This is less secure than using digest authentication, but may be required by some clients. ■ Remove **axia.digest.auth** altogether. This disables security between the Administration Server and client. This is only appropriate for test and evaluation deployment deployments used by trusted personnel. The default value is **axia.digest.auth=true**. Set in **admin.properties**.

Table A–9 (Cont.) System Security Properties

System Security Property	Description
axia.console.password.validation.enabled	Boolean. Enables/disables password strength validation. If **true**, the restrictions in **axia.console.password.validation.min_length**, a**xia.console.password.validation.min_length**, **axia.console.password.validation.require_lower**, a**xia.console.password.validation.require_upper**, and **axia.console.password.validation.require_digit** are enforced. Default value is **true**. Set in **common.properties**.
axia.console.password.validation.min_length	Defines the minimum password length. Enforced if **axia.console.password.validation.enabled** is set to **true**. Default value is 6 characters. Set in **common.properties**.
axia.console.password.validation.require_lower	Boolean. Enables/disables requirement that passwords include at least one lower-case character. Enforced if **axia.console.password.validation.enabled** is set to **true**. Default is **true**. Set in **common.properties**.
axia.console.password.validation.require_upper	Boolean. Enables/disables requirement that passwords include at least one upper-case character. Enforced if **axia.console.password.validation.enabled** is set to **true**. Default value is **true**. Set in **common.properties**.
axia.console.password.validation.require_digit	Boolean. Enables/disables requirement that passwords include at least one digit. Enforced if **axia.console.password.validation.enabled** is set to **true**. Default value is true Set in **common.properties**.
axia.digest.auth	Boolean. Specifies whether to use digest access authentication when the Administration Console connects to the Administration Server. Set this property to: ■ **true** to use HTTP digest authentication. ■ **false** to not use HTTP digest authentication. The default value is **false**. Set in **admin.properties**.
axia.ssl	Boolean. There are two of these settings and the default value for both is **true**. One is in the **common.properties** file that controls whether the Administration Console is required to use SSL security for all traffic. The other is the master SSL switch for the managed server. If **false**, no traffic with the managed server is required to use SSL security. If **true**, SSL security is required.

Table A–9 (Cont.) System Security Properties

System Security Property	Description
axia.ssl.cipher_suites	Specifies the combinations of ciphers that Service Broker supports for SSL communication between the Administration Server and its clients. The choices are: ▪ **TLS_RSA_WITH_AES_128_CBC_SHA** ▪ **TLS_DHE_RSA_WITH_AES_128_CBC_SHA** ▪ **TLS_DHE_DSS_WITH_AES_128_CBC_SHA**
https.cipherSuites	Specifies the combinations of ciphers that Service Broker supports for HTTPS communication between the Administration Server and its clients. The choices are: ▪ **TLS_RSA_WITH_AES_128_CBC_SHA** ▪ **TLS_DHE_RSA_WITH_AES_128_CBC_SHA** ▪ **TLS_DHE_DSS_WITH_AES_128_CBC_SHA**
javax.net.ssl.keyStore	The file name of the keystore to use for Processing Servers, Signaling Servers and administration tools. The keystore is a file that contains public and private keys used to establish SSL connections. Set in **common.properties** for the administration tools. Set in **server.properties** for the Processing Server and the Signaling Server.
javax.net.ssl.trustStore	The file name of the truststore to use for Processing Servers, Signaling Servers and administration tools. The truststore is a file that contains public certificates used to establish SSL connections. Set in **common.properties** for the administration tools. Set in **server.properties** for the Processing Server and the Signaling Server.
org.eclipse.equinox.http.jetty.ssl.keystore	Specifies the keystore to use for the Jetty HTTPS connection between the Administration Console and the Administration Server. This entry is commented-out by default. If not specified, the same keystore as defined in the property **javax.net.ssl.keyStore** is used. Set in **admin.properties**. and **hosting.properties**.

Directory Contents and Structure for a Domain Configuration

Table A–10 gives information about the directory structure and contents of a domain configuration.

Table A–10 Directory Structure for a Domain Configuration

Directory	Description
Domain_home	Top-level directory for a domain configuration. This directory contains: ▪ **initial.zip** Contains references to all modules for Processing Servers and Signaling Servers. ▪ **modules** A directory with OSGi bundles deployed on the Processing Servers and Signaling Servers in the domain. ▪ **admin_lock.dat** Lock file used to ensure exclusive write-access to the domain configuration.
Domain_home/**modules**	Contains binaries and configuration data for Processing Servers and Signaling Servers in the domain.

Directory Structure and Contents for JDKs

A bundled JDK can be installed when an administration client, a Processing Server, and a Signaling Server are installed.

These files are located in under the directory:

Linux and Solaris: *Oracle_home*/**ocsb61**

Oracle_home is the Oracle home directory you defined when you installed the product.

Table A–11 describes the directory structure and the contents of the directory structure.

Table A–11 Directory Structure for JDKs Relative to Oracle_home/ocsb61

Directory	Description
jdk*version*	Contains Sun HotSpot JDK. *version* correlates to the version of the JDK, for example **1.6.0_14** This directory is created only if you specified to install Sun HotSpot JDK during the installation.
jrrt-*version*	Contains Oracle JRockit JDK. *version* correlates to the version of the JDK, for example **3.1.0-1.6.0** This directory is created only if you specified to install Oracle JRockit JDK during the installation.

Directory Structure and Contents for Oracle Universal Installer

A set of files and directories are created by Oracle Universal Installer.

These files are located under the directory:

Oracle_home/**ocsb61**

Oracle_home is the Oracle home directory you defined when you installed the product.

Table A–12 describes the directory structure and the contents of the directory structure.

Table A–12 Directory Structure for Oracle Universal Installer Relative to Oracle_home/ocsb61

Directory	Description
cfgtoollogs	Contains log-files related to Oracle Universal Installer.
inventory	Contains inventory files maintained by Oracle Universal Installer.

Safe Services

Safe services is a set of services that are installed and running when the platform is in state SAFE MODE. They are the bare minimum of services that needs to be running in order to fetch server services, applications, and protocol adapters for the domain configuration and start them. Table A–13 lists these services.

See "Life Cycle of Processing Servers and Signaling Servers" for details on SAFE MODE.

Table A–13 Safe Services

Service	OSGi Bundles
Provisioning service	oracle.axia.platform.provisioningservice
Logging-related	com.bea.core.apache.log4j oracle.axia.platform.loggingservice
Services related to Equinox OSGi Framework	org.eclipse.osgi.services org.eclipse.osgi.services org.eclipse.equinox.ds org.eclipse.equinox.util