Oracle® Communications Service Broker

System Administrator's Guide

Release 6.0

E23523-02

March 2012

ORACLE®

Oracle Communications Service Broker System Administrator's Guide, Release 6.0

E23523-02

Contents

12 Setting Up Servers in Signaling and Processing Domains

13 Starting and Stopping Processing Servers and Signaling Servers

14 Starting and Stopping the SS7 Process

15 Upgrading and Patching Service Broker

16 Maintaining Oracle Communications Service Broker

17 Life Cycle of Processing Servers and Signaling Servers

18 Monitoring Service Broker

Preface

This document describes how to:

- Start and stop Oracle Communications Service Broker
- Add and remove servers from a domain
- Maintain your installation
- Upgrade and patch servers
- Use scripts for automation purposes

It also has a reference appendix, with information about directory structures and content, environment variables, and system properties.

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

Audience

This document is intended for system administrators, developers, and system integrators who want to run Oracle Communications Service Broker.

This documentation is based on the assumption that readers are already familiar with:

- The operating system on which your system is installed
- Telecom networks and protocols
- Clustering concepts and life cycle management
- Java
- 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 6.0 documentation set:

- *Oracle Communications Service Broker Release 6.0 Release Notes*
- *Oracle Communications Service Broker Release 6.0 Concepts Guide*
- *Oracle Communications Service Broker Release 6.0 Configuration Guide*
- *Oracle Communications Service Broker Release 6.0 System Administrator's Guide*
- *Oracle Communications Service Broker Release 6.0 Integration 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 main concepts that you need to understand before you begin configuring Oracle Communications Service Broker. In addition, the chapter explains tools that you can use to configure Service Broker.

Configuration Overview

A Service Broker deployment includes two types of domains:

- Signaling Domain: Used to manage the servers of the Signaling Tier and the SSUs running on these servers.
- Processing Domain: Used to manage the servers of the Processing Tier and the module instances (that is OE and IM instances) running on these servers.

Each domain has an associated domain configuration, which is stored in the domain configuration directory. When you make configuration updates, you need to make changes in the domain configuration directory.

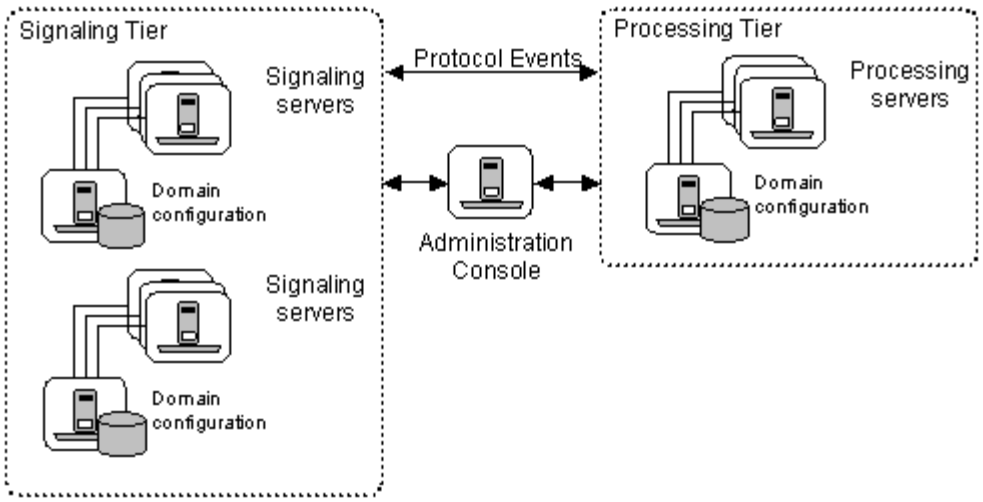
About Domains

A set of Processing Servers are grouped into a Processing Domain, and a set of Signaling Servers are grouped into a Signaling Domain.

Servers within a domain are symmetrical. This means that all servers in a domain have the same bundles deployed and started. Each domain has an associated domain configuration located in a directory that the domain servers can access. Servers can access the domain configuration directory either using a shared file system or through HTTP.

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

Figure 1–1 Overview of Domains and Tiers



About Domain Configuration Modes

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

- Online mode, when configuration updates are propagated to all servers in the domain as the changes are carried out.
- Offline mode, when updates are carried out only to the domain configuration and applied to each server when it is re-started.

  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 doing a rolling upgrade of an installation.

About Locking Domain Configuration for Changes

You need to lock a domain configuration in order to make configuration updates to it. When you lock the domain, it is locked for edits from other administration clients.

After locking a domain configuration, you can make changes to it. You need to commit the changes in order to apply them on the domain configuration.

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

About Configuration Tools

You can access a domain configuration directory using one of the following tools:

- Administration Console: Graphical user interface.
- JMX Configuration MBeans: Java software API based on standard Java Management Extensions (JMX). The API provides a system interface for configuration.
- Scripting Engine: JMX Configuration MBeans are also accessible through a Scripting Engine. The script format is an XML which represent MBeans and MBean attributes that you want to configure. See “Using Scripting for

Configuration and Management" in *Oracle Communications Service Broker System Administrator's Guide* for instructions.

- JMX-compliant GUI: Any JMX-compliant GUI can be used, for example, JConsole, which is included with JDK.

All configuration tasks can be achieved in either way. You can use the Administration Console, the Configuration MBeans, the Scripting Engine and the JMX-compliant GUI interchangeably to make configuration changes, because all tools use Configuration MBeans as their underlying layer.

About the Administration Console

You can manage a domain configuration using the Administration Console. It renders in the GUI the data stored in the domain configuration directory, and it has read and write access to the domain configuration directory.

The Administration Console can be installed and run from any system that has access to the domain configuration directory.

The Administration Console can run in two ways:

- Standalone Administration Console, which is a desktop software application provided with Service Broker.
- Web Administration Console, which you can access using a standard web browser.

The Administration Console manages a single domain. In a typical production deployment there are two instances of the Administration Console, one to manage the Signaling Domain and another to manage the Processing Domain.

Note: In a test environment, where one domain includes both Signaling Servers and Processing Servers, there is only one Administration Console instance. In this case, the Administration Console manages both the Signaling Tier and Processing Tier in one domain.

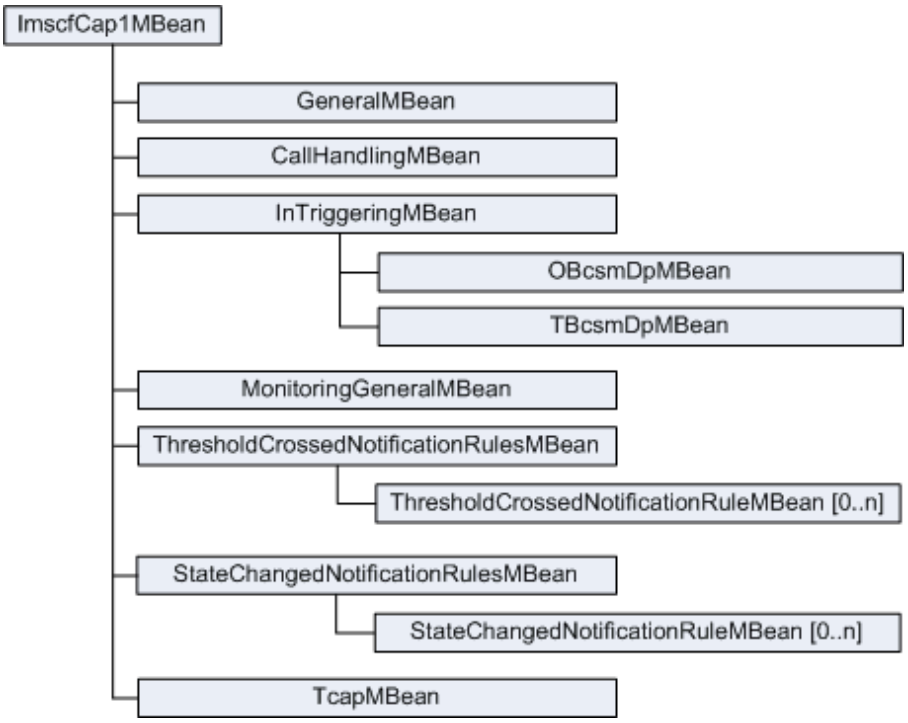
About Configuration MBeans

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 system resources (for example, applications, devices). The resources are represented by objects called Management Beans (MBeans). JMX configuration MBeans are simple Java objects that provide an API to set/get configuration attributes. Service Broker configuration is entirely supported by JMX configuration MBeans and can be used by JMX clients to set/get Service Broker configuration. JMX is specified in the Java Specification Requests.

You can access and manage a domain configuration using the configuration MBeans on the domain's Administration Server (where the domain Administration Console is running). Each component in a domain, that is SSU, OE, IM or SM, has a set of MBeans, organized in a logical hierarchy, that together form the complete component configuration.

Figure 1–2 shows an example of the MBean hierarchy for the IM-SCF CAP phase 1 component:

Figure 1–2 Example of an IM Configuration MBean Hierarchy



Service Broker MBean Object Names

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

```
com.convergin:Type=MBean-type-name,Target=server,Version=version_number,
Location=AdminServer,Name=component-name.unique-resource-name
```

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

Table 1–1 Service Broker MBean Object Name Key Properties

Property	Description
Type=*MBean-type-name*	The short name of the MBean's type. The short name is the unqualified type name without the MBean suffix. For example, for an MBean that is an instance of the LinksetMBean, the Type would be Linkset.
Target=*server*	In the specific case of SS7 SSU MBeans, this property specifies the name of the target server where the SSU is running.
Version=*version_number*	Specifies the version of the MBean instance. When you upgrade an MBean to a later version, this parameter 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.
Location=AdminServer	Specifies the location of an MBean. This parameter is always set to AdminServer.

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

Property	Description
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.

For example:

```
com.convergin:Type=IpRemoteSystem,Target=ssu_managed_1,Version=1.0,
Location=AdminServer,Name=sbssuss7.SG01
```

About JConsole

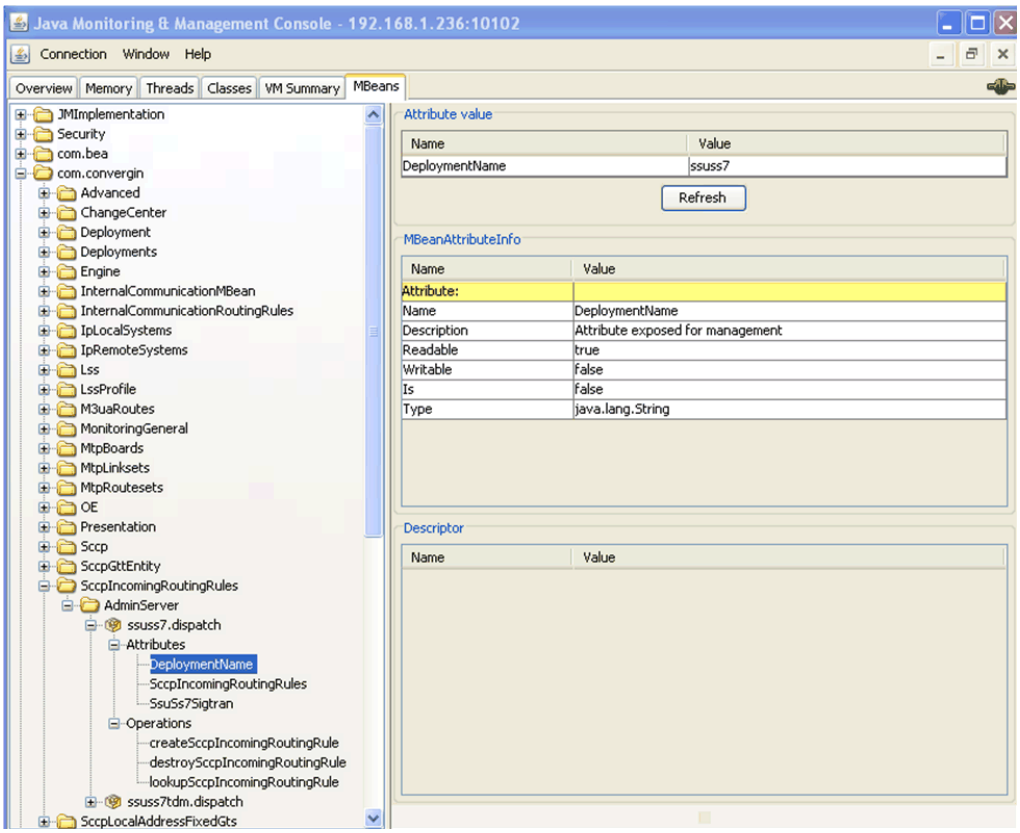
JConsole is a Java Monitoring and Management Console included in JDK 5.0 or higher. It provides configuration GUI for applications running on java platforms and supporting the Java Management eXtension technology. By using JConsole you can manage the Service Broker configuration MBeans.

With JConsole you can:

- 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

Figure 1–3 shows an example of using JConsole for viewing information about an MBean instance's attribute.

Figure 1–3 Viewing Information about MBean Instance's Attribute



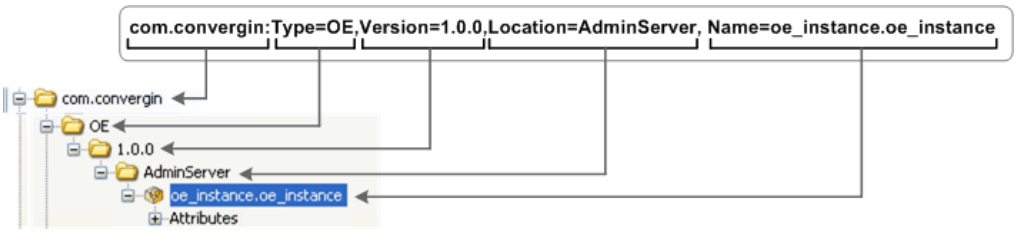
For more information on using JConsole, see http://docs.oracle.com/javase/6/docs/technotes/guides/management/jconsole.html

Understanding the Hierarchy of MBeans in JConsole

For each individual MBean, JConsole automatically generates a tree structure based on the object name of an MBean. See "Service Broker MBean Object Names" for more information about the format of MBean object names.

In this tree structure, each nested node represents a single property of the object name. For example, the object name of an OeMbean instance is com.convergin:Type=OE,Version=1.0.0,Location=AdminServer,Name=oe_instance.oe_instance. Figure 1–4 shows the tree structure that JConsole generated for this MBean.

Figure 1–4 MBean Tree Structure

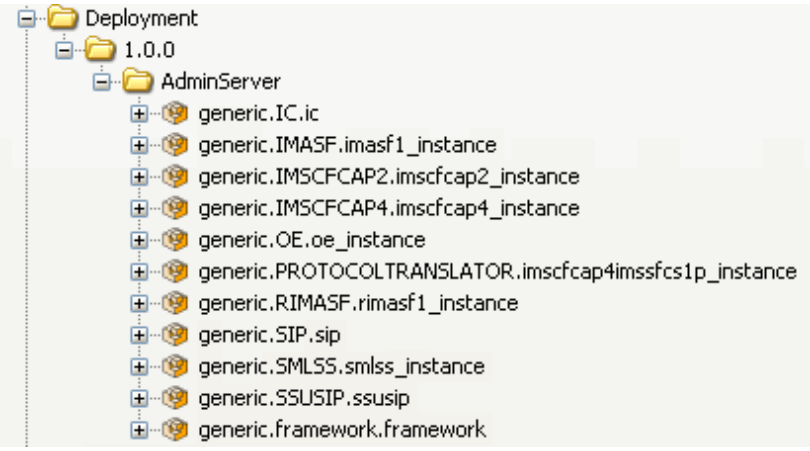


Notice that all Service Broker MBeans are located under the com.convergin node, because they all have a com.convergin domain in their object name. Under the

com.convergin node you'll find all the types of Service Broker MBeans, each node represents one type of MBean.

Different instances of the same MBean type are displayed as different nodes nested into the node representing their common type. For example, Figure 1–5 shows different instances of DeploymentMBean displayed as different nodes under the Deployment node.

Figure 1–5 Example of Displaying Different Instances of the Same Type



About the Scripting Engine

You can manage and configure Service Broker programatically using scripts run by the Scripting Engine. The Scripting Engine has an MBean server that you can use to manage Service Broker configuration MBeans.

You define MBean operations to run and/or MBean parameters to change in your scripts and then execute them from the Scripting Engine. The script format is an XML representation of the MBeans.

See "Using Scripts for Configuration and Management" for details.

2

Using the Administration Console to Configure a Domain

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

About the Administration Console

The Administration Console provides a graphical user interface to perform configuration tasks easily and intuitively, without programming knowledge. For example, using the Administration Console you can add servers to a domain, configure the Signaling Domain, deploy interworking modules in the Processing Domains, and configure the Orchestration Engine.

The Administration Console manages a single domain. In a typical production deployment there are two instances of the Administration Console, one to manage the Signaling Domain and another to manage the Processing Domain.

Service Broker provides the following types of the Administration Console:

- A Standalone Administration Console, which is a desktop software application provided with Service Broker.
- A Web Administration Console, which you access using a standard web browser.

The Standalone and Web Administration Consoles have very similar user interfaces.

Understanding the Administration Console User Interface

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

Table 2–1 Administration Console Working Areas

Working Area	Description
Domain Navigation pane	Provides a hierarchical view of the domain. You use the Domain Navigation pane 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 OE, IMs, and Subscriber Store settings. The Navigation pane also includes Domain Management settings.
Configuration pane	Displays configuration parameters of components that you select in the Domain Navigation pane. You use the Configuration pane to set up and modify configuration of components deployed in the domain.

Table 2–1 (Cont.) Administration Console Working Areas

Working Area	Description
Change Center pane	You can manage changes that you make in the configuration of domain's components. See "Making Configuration Changes in a Domain" for more information.

Making Configuration Changes in a Domain

When 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 one transaction and applies all these changes at once.

You can perform the following actions when updating domain configuration:

- Locking Domain for Changes (see "Locking Domain for Changes" for more information)
- Committing Changes (see "Committing Changes" for more information)
- Discarding Changes (see "Discarding Changes" for more information)

Locking Domain for Changes

To make changes in the domain configuration, you need to lock this domain for changes. After you locked the domain, no one else can do any changes in the domain. During the lock, Service Broker accumulates all changes. However, Service Broker does not apply these changes unless you commit them.

To lock a domain for changes:

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

Committing Changes

You need to commit the changes in order to apply them. Service Broker applies all changes that you have done after locking the domain for changes.

To commit changes:

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

Discarding Changes

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

To discard changes:

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

Switching the Domain Configuration Mode

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

- Online mode, when configuration updates are propagated to all servers in the domain as the changes are carried out.

- Offline mode, when updates are carried out only to the domain configuration and applied to each server when it is re-started, the domain configuration is offline.

  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 doing a rolling upgrade of an installation.

To switch between the domain configuration 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 Java MBeans to Configure a Domain

This chapter describes how to configure a Service Broker domain using Java MBeans.

Opening a Domain for Configuration

To make changes in the domain configuration, you need to open this domain for configuration. After you opened the domain for configuration, no one else can do any changes in the domain.

To open a domain for configuration:

- Invoke the following operation of DomainServiceMBean:

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

See "DomainServiceMBean" for more information on this MBean.

Making Configuration Changes in a Domain

After you opened a domain for configuration, you can make configuration changes in the following modes:

- Autocommit mode

  When you update configurations in this mode, changes are committed and written to the configuration directory immediately. This is the default configuration mode.

- Transaction mode

  When you update configuration in this mode, multiple changes accumulate into one transaction. Setting the domain configuration to transaction mode makes it possible to perform a set of configuration updates and have them applied all at once.

  You can perform the following actions in the transaction mode:

  - Setting the transaction mode (see "Setting the Transaction Mode" for more information)
  - Discarding changes (see "Discarding Changes" for more information)
  - Committing accumulated changes (see "Committing Changes" for more information)

Setting the Transaction Mode

To make changes in the transaction mode:

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

  ```
  void begin()
  ```

See "ConfigurationAdminMBean" for more information on this MBean.

Discarding Changes

To discard the accumulated changes:

- Invoke the following operation of ConfigurationAdminMBean:

  ```
  void rollback()
  ```

See "ConfigurationAdminMBean" for more information on this MBean.

Committing Changes

To commit the accumulated changes:

- Invoke the following operation of ConfigurationAdminMBean:

  ```
  void commit()
  ```

See "ConfigurationAdminMBean" for more information on this MBean.

Releasing the Lock

To release the lock:

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

  ```
  void closeDomain()
  ```

See "DomainServiceMBean" for more information on this MBean.

Specifying the Domain Configuration Mode

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

- Online mode, when configuration updates are propagated to all servers in the domain as the changes are carried out.
- Offline mode, when updates are carried out only to the domain configuration and applied to each server when it is re-started.

  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 doing a rolling upgrade of an installation.

To specify the domain configuration mode:

- Set the **OnLine** attribute of **DomainServiceMBean** to one of the following values:
  - Online
  - Offline

Managing Domain Properties

Each domain configuration has a set of properties. The properties are name-value pairs. See Table 3–1.

The domain properties are set during domain creation. After the domain is created, you can change domain properties at any time using **DomainServiceMBean**.

You specify the domain properties during the domain creation properties.

Table 3–1 Domain Properties

Name	Value
axia.domain.host	Specifies the URI to the domain configuration. Format: [**file** | **http**]**://***Context_path* If your domain configuration is accessed using the Domain Web server, use the scheme **http://**. Example: **http://myhost:9000/** If your domain configuration is accessed using a shared file system, use the scheme **file://**.
axia.ssl	Specifies if SSL is enabled or disabled for management operations. Set this property to: ■ **true** to enable SLL ■ **false** to disable SSL See "Configuring Security Between Service Broker Components".
axia.domain.id	Specifies the domain name in a multi-processing domain. To maintain domain exclusivity, each domain is assigned a unique name. All domains with the same ID are assumed to be in the same domain. Format: Can only contain letters, digits or underscores (a-z, A-Z, 0-9_). Case sensitive. Length: Between 1 and 8 characters. Examples: ■ 23DoMain ■ domain_1 If no name is specifically assigned, the value is **default**.

To change or set a domain property, invoke the operation **setDomainProperty** on the MBean **DomainServiceMBean**.

DomainServiceMBean

JAR File

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

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

Package

oracle.axia.api.management.ds

Object Name

oracle:type=oracle.axia.api.management.ds.DomainServiceMBean

Factory Method

Created automatically.

Attributes

None.

Operations

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

Adds a new server to a domain configuration.

Parameters:

- **name** Name of the server.
- **host** Host of the server.
- **port** General purpose port of the server.
- **adminPort** Administration port of the server.
- **jmxJrmpPort** JMX port of the server.
- **jmxRegistryPort** JMX registry port of the server.

void closeDomain()
Closes a domain that has been opened for updates.

void createDomain(String type, String domainPath)
Creates a domain that is accessed by the servers using a shared file system.

Parameters:

- **type** Type of domain to create. See the discussion on creating the domain in *Service Broker Installation Guide* for available options.
- **domainPath** Path to the directory where the domain configuration is created.

void createHostedDomain(String type, String domainPath, String hostAddress)
Creates a domain that is accessed by the Service Broker servers using HTTP or HTTPS.

Parameters:

- **type** See **createDomain**.
- **domainPath** See **createDomain**.
- **hostAddress** Host and port for the Domain Web Server.

void editManagedServer(String name, String host, int port, int adminPort, int jmxJrmpPort, int jmxRegistryPort)
Edits the settings for a server.

Parameters:

- **name** Name of the server.
- **host** Host of the server.
- **port** Port of the server.
- **adminPort** Administration port of the server.
- **jmxJrmpPort** JMX port of the server.
- **jmxRegistryPort** JMX registry port of the server.

String getDomainProperty(String name)
Gets the value of a domain property. See Table 3–1.

Parameter:

name Name of the property.

isOffline()
Returns true if the domain is in offline configuration mode.

java.util.List<String>listDomainTypes()
Lists available domain types.

java.util.List<String>listServers()
Lists the name of the servers in the domain.

void openDomain(String domainPath)
Opens a domain for editing.

Parameter:

domainPath Path to the directory where the domain configuration is located.

void removeManagedServer(String name)
Removes a server from a domain.

Parameter:

name Name of the server to remove.

void renameDomainProperty(String oldName, String newName, String value)
Renames a domain property. See Table 3–1.

Parameters:

- **oldName** Old name of the property.
- **newName** new name of the property.
- **value** Value of the property.

void setDomainProperty(String name, String value)
Sets a property for a domain. See Table 3–1.

Parameters:

- **name** Name of the property.
- **value** Value of the property.

void setOffline(boolean offline)
Sets the domain to Offline or Online mode.

Parameter:

offline True to set the domain configuration mode to offline, and false to set it to online.

ConfigurationAdminMBean

JAR File

oracle.axia.cm.api-*version*.jar

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

Package

oracle.axia.api.management.cm

Object Name

oracle:type=oracle.axia.api.management.cm.ConfigurationAdminMBean

Factory Method

Created automatically.

Attributes

None.

Operations

void begin()
Sets the configuration changes mode to the Transaction mode.

void commit()
Commits configuration changes accumulated since the configuration changes mode changed to the Transaction mode using the begin operation.

boolean isTransactionActive()
Indicates if the configuration changes mode is set to the Transaction mode.

java.util.List<String>listPendingConfiguration()
Lists pending configuration entries accumulated since the configuration changes mode changed to the Transaction mode using the begin operation.

void rollback()
Discards any pending configuration entries accumulated since the configuration changes mode changed to the Transaction mode using the begin operation.

4

Configuring Service Broker Using JConsole

This chapter describes how to start the Service Broker JConsole.

Starting JConsole

To start JConsole:

1. Run **jconsole.exe**.

   jconsole.exe is located in JDK_HOME/bin, where JDK_HOME is the directory in which the Java Development Kit (JDK) is installed.

   The JConsole: New Connection dialog box appears.

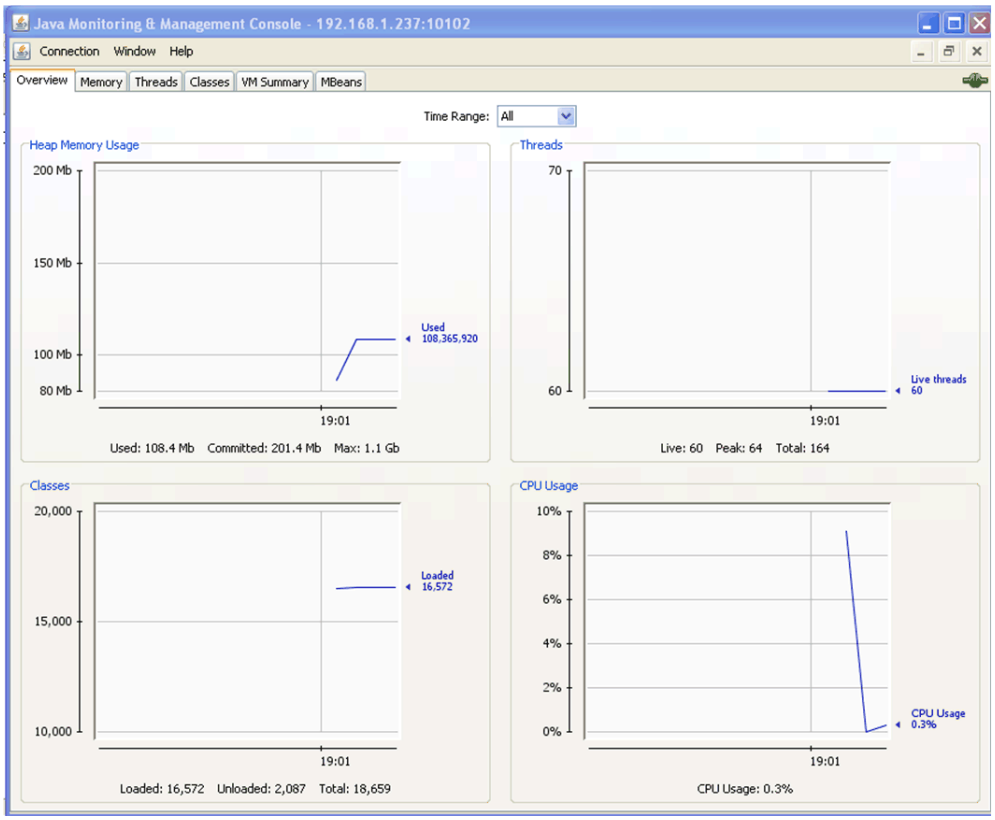
   Figure 4–1 shows the JConsole: New Connection dialog box where you can enter the IP address of the Administration Console Server to which you want to connect.

Figure 4–1 JConsole: New Connection

2. Select the **Remote Process** button and specify the IP address and port of the Administration Console Server on which configuration MBeans run.
3. Click **Connect**.

   The Java Monitoring and Management Console window appears. Figure 4–2 shows the JConsole Overview tab.

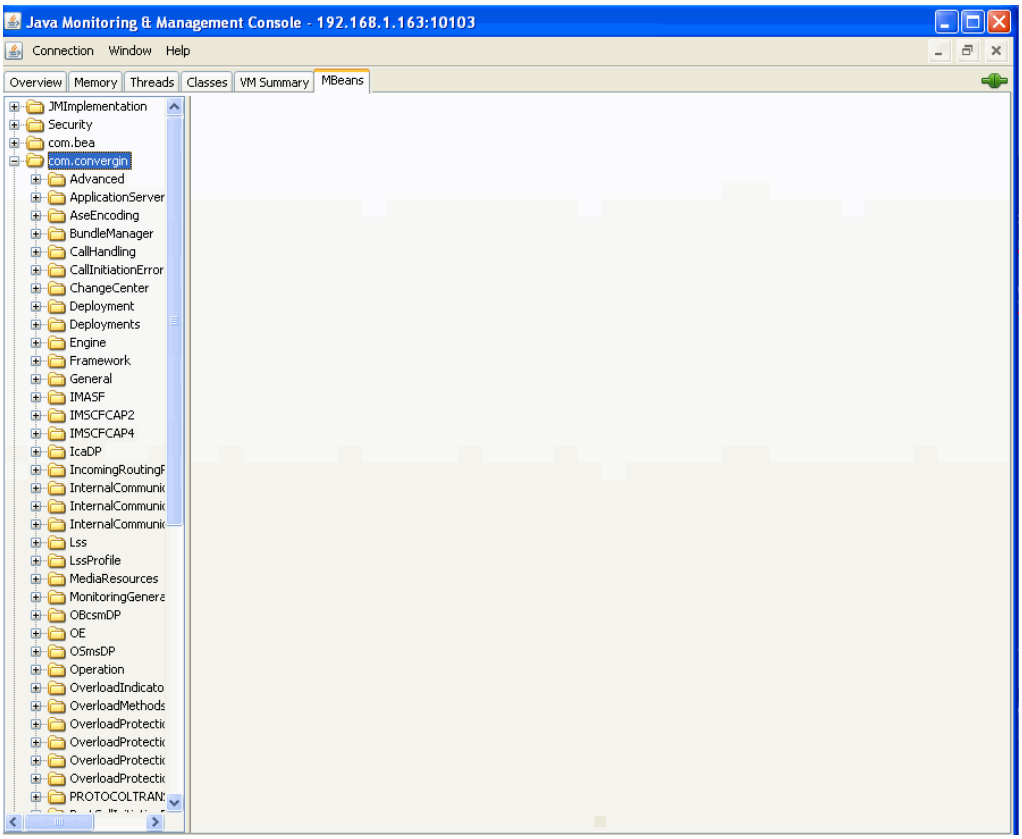
Figure 4–2 Java Monitoring and Management Console



4. Click the **MBeans** tab.

   A list of available domains and their MBeans appears in the left pane. Figure 4–3 shows the JConsole MBeans tab that contains Service Broker configuration MBeans.

Figure 4–3 MBeans Tab



5. In the navigation tree, expand the **com.convergin** node.

   This node contains all Service Broker MBeans.

6. Under com.convergin, select the MBean that you want to monitor.

   The information about the selected MBean is displayed in the right pane.

5

Using Scripts for Configuration and Management

This chapter describes how to use scripts to configure and manage your deployment.

Understanding Scripting

You can use scripts to manage and configure Service Broker.

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.

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 5–1 describes the syntax of the script file.

Table 5–1 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 MBeans JavaDoc 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.
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 of Complex Data Structures" for complex data types.

Table 5–1 (Cont.) Structure of an XML Management Script

Element	Description
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 5–1 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 5–1 Getting an MBean Attribute

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

Example 5–2 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 5–2 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 5–3 describes how to invoke the operation with the signature **void openDomain(java.lang.String domainPath)** with **domainPath** set to **/usr/local/sb/domain**.

Example 5–3 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 of 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 5–4 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 5–4 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***/ocsb60/admin_console**.

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.

6

Configuring Security Between Service Broker Components

This chapter describes the security model for Oracle Communications Service Broker and explains how to configure it.

About the Service Broker Security Model

See the *Service Broker Security Guide* for an overview of the Service Broker security model and philosophy. This chapter explains the details of setting up security for your Service Broker implementation.

Briefly, Service Broker The security model is based on Secure Socket Layer (SSL) and files system-level access privileges.

Securing Administration Clients, Processing Servers, and Signaling Servers

The Processing Servers and Signaling Servers expose Java MBeans. There is an MBean server in each of these servers. The MBean server is using a JMX SSL connection that requires both the server and an administration client to authenticate. Administration clients includes:

- Standalone Administration Console
- Web Administration Console server
- Scripting Engine

The same security mechanisms that applies to these tools also applies to any administration client that uses JMX to configure, administer, and operate Service Broker.

Oracle recommends that you secure the JMX port with SSL encryption to enable client authentication. SSL encryption is enabled by default and is controlled by using entries in the *Oracle_home***/ocsb60/admin_console/properties/common.properties** file. See Table A–9 in System Administrator's Reference for details on these entries.

SSL encryption requires a public key infrastructure (PKI) where each server has a public key and a private key. The key-pair is stored in an entry in a keystore. The private key is used by the server itself. The public key is used by the administration clients.

Each administration client also has a private key and a public key.

A public key is wrapped in a certificate, a public certificate. A certificate can be either self-signed or issued by a Certificate Authority (CA).

Each server has two stores:

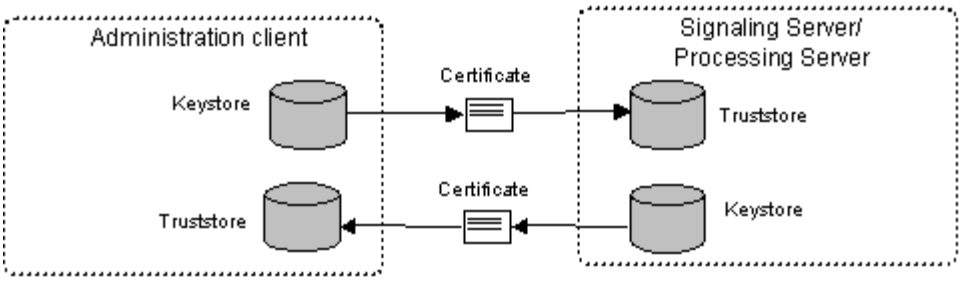
- Server keystore
- Administration client truststore

Each administration client has two stores:

- Administration client keystore
- Server truststore

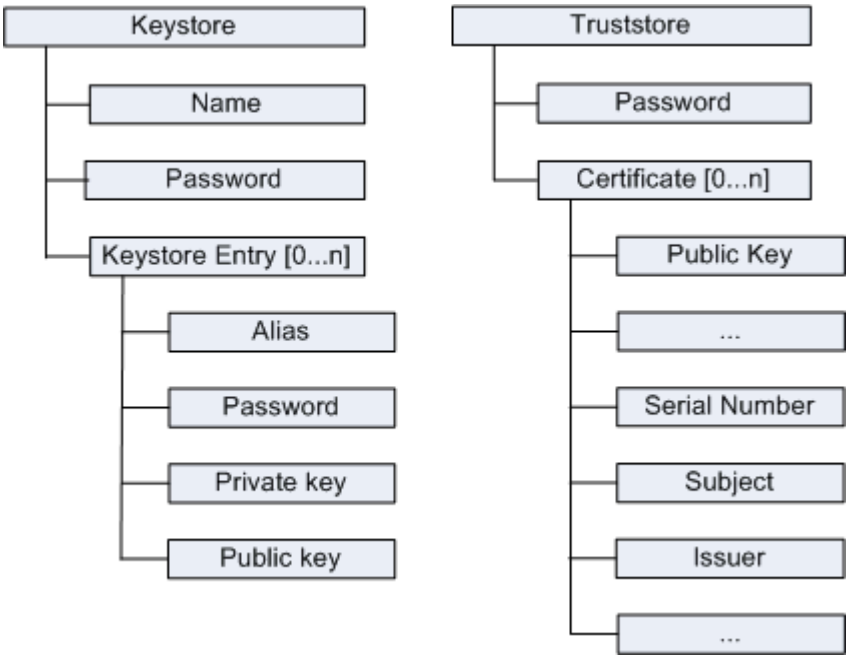
Figure 6–1 shows the relationship between the server stores and the administration stores.

Figure 6–1 Keystores, Truststores, and Exchange of Certificates



A keystore contains keystore entries. A truststore contains public certificates. Figure 6–2 shows the structure of the keystore and truststore.

Figure 6–2 Keystores and truststores



The keystores, truststores, and certificates are files. Certificates are exported from keystores, or provided by CAs, and imported to truststores.

Each keystore has a name and a password. Each keystore entry has an alias that identifies a key-pair and a password for the entry.

Each truststore has a password. A truststore can contain one or more certificate.

The certificate contains the public key and data about the certificate such as serial number, subject (the entity being identified by the certificate), and issuer of the certificate.

It is possible to use the same certificate for all administration clients, meaning that the server truststore contains only one administration client certificate. This approach is less secure than having individual certificates for each administration client.

Domain Configuration Security

File system-level security mechanisms are used for controlling access to the domain configuration. The user that starts any of the following administration clients must have read and write privileges to the domain configuration directory and all files in it:

- Standalone Administration Console
- Web Administration Console server
- Scripting Engine

The Domain Web server must have read access to the domain configuration directory and all files in it.

Use operating-system specific commands to:

- Configure users that have privilege to start the administration clients.
- Make sure these users have read-write or read access to the domain configuration directory.

Oracle recommends that you only allow the owner or a trusted user group to have any privileges to the directory.

Securing Hosted Domains

When using a hosted domain configuration that is accessed using a Domain Web server, you can choose to access it using HTTP or HTTPS. The default is to use HTTPS with SSL client certificate authentication.

These settings are controlled by entries in the *Oracle_home***/ocsb60/admin_console/properties/hosting.properties** file. See Table A–3 in System Administrator's Reference for details on these entries.

When HTTPS and client authentication are enabled, you must configure the trusted client certificates on the managed servers in your domain. You have these options for storing the client certificates:

- Import them into default domain hosting keystore specified by the **javax.net.ssl.keyStore** parameter in **common.properties** (**clientkeystore** is the default). This is somewhat counter-intuitive. The Service Broker hosted domain reads trusted client authentication certificates from its *client* keystore.
- Import them into a different keystore that you specify by uncommenting the **org.eclipse.equinox.http.jetty.ssl.keystore=jettysslkeystore** entry in the **hosting.properties** file. You can either use the default **jettysslkeystore**, or specify a different keystore.

Securing the Web Administration Console-Server Connection

An HTTPS connection is used between the Web Administration Console and the Web Administration Console server.

The first time the Web Administration Console is accessed, you are prompted to accept the certificate provided in a keystore in the Web Administration Console server. How you are prompted depends on:

- The Web browser you use
- If a self-signed certificate is used or if the certificate was provided by a certificate authority

The Web Administration Console user is required to authenticate with the Web Administration Console server, using the HTTP Basic Authentication mechanism.

The authentication requires the user to enter a user name and password in the browser. The user name and password is provided when the Web Administration Console server is started and the same credentials are used when the Web Administration Console is accessed.

Securing the Administration Port

Configuration updates and deployment updates are propagated to Processing Servers and Signaling Servers over the administration port defined for the server.

Oracle recommends that you secure the port with SSL encryption and to have client authentication enabled. Set the **axia.ssl** domain properties to **true**.

The property **axia.admin.verify.hostname** specifies if hostname verification is used for the SSL connection between the Administration Console and the servers. If set to **true**, each server must specify a host identity that matches the managed server listening address as specified in its key.

See "Enabling and Disabling SSL".

There are also requirements on how the SSL certificates are generated:

- If the server name is specified as a host name in the domain configuration, one of the following applies as to how the common name (CN) in the certificate for the server is specified:
  - The CN in the certificate for the server is identical to the server name defined in the domain configuration.

    In this case, a certificate must be created for each server.
  - The CN in the certificate is set to **.domain*.

    In this case, all servers can use the same certificate.

    This requires that the server names must be specified with their full names, including the domain, in the domain configuration. For example if the servers names are **server1.us.oracle.com**, **server2.us.oracle.com**, and so on, the CN is set to ***.us.oracle.com**.
- If the server name is specified as an IP-address in the domain configuration, a certificate must be generated for each server.

For more information on server identity certificate validation in HTTP/TLS, see section *3.1. Server Identity* in RFC 2818:

`http://www.ietf.org/rfc/rfc2818.txt`

Setting Service Broker Console Password Strength Validation

By default, all passwords used for the Web Administration Console and the Domain Web server must be at least six characters long, contain at least one lower case character, one upper case character, and one digit.

You have the option of changing these password strength requirements using property in the *Oracle_home***/ocsb60/admin_console/properties/common.properties** file:

- Whether the password is validated.
- A minimum length for the password
- Whether an upper-case character is required in the password.
- Whether a lower-case character is requried in the password.
- Whether an integer is requried in the password.

When you change these properties, you need to restart all Processing Servers, Signaling Servers, the Web Administration Console server, and the Domain Web server. SSL is enabled or disabled on deployment level, so all servers and administration clients must have the same setting.

See Table A–10 in "System Administrator's Reference" for a details on these properties.

Enabling and Disabling SSL

Note: Oracle recommends that you do not disable SSL.

The Java system property **axia.ssl** specifies if SSL is enabled or disabled. By default, SSL is enabled.

If SSL is enabled, use HTTPS to connect to the Web Administration Console server or Domain Web server. If SSL is disabled, use HTTP.

The Java system property **axia.ssl** specifies if SSL is enabled or disabled for the administration port. Configuration updates and deployment operations are propagated to Processing Servers and Signaling Servers over this port.

The property **axia.ssl.cipher_suites** specifies the enabled platform SSL cipher suites. See *Java Cryptography Architecture Sun Providers Documentation* Cipher Suite documentation for Sun JSSE Provider for supported values, including how to use unlimited encryption options. The property is set to a comma-separated list of cipher suites.

For a complete list of the domain properties see "System Administrator's Reference".

When you change these properties, you need to restart all Processing Servers, Signaling Servers, the Web Administration Console server, and the Domain Web server. SSL is enabled or disabled on deployment level, so all servers and administration clients must have the same setting.

Defining Keystores and Truststores

The default keystore and the truststore for each server and administration client are located in these directories:

*Oracle_home***/ocsb60/managed_server**

*Oracle_home***/ocsb60/admin_console**

The default keystore name is **keystore**. The default truststore is **truststore**. You can change the names and location of the stores. The Java system property:

- **javax.net.ssl.keyStore** specifies the location and name of the keystore.
- **javax.net.ssl.trustStore** specifies the location and name of the truststore.

See "System Properties" in "System Administrator's Reference" for details.

Setting up Public Key Infrastructures for Service Broker Components

This section provides an overview of the keytool program and then explains how to set up the Service Broker public Key infrastructure (PKI). Service Broker uses the PKI to store the credentials required to communicate between administration clients, Processing Servers, and Signaling Servers.

About Keytool and X.500 Distinguished Names

You set up the Service Broker PKI using **keytool**, a key and a certificate management tool. keytool is a part of the Oracle Java Development Kit (JDK) and is located in the *Oracle_home***/ocsb60/jdk/bin** by default. See:

`http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html`

A Distinguished Name (DN) identifies an entity. A DN is used when generating X.509 certificates. keytool uses a string notation for DN, immediately following the **-dname** parameter.

CN=*common_name***, OU=***organization_unit***, O=***organization_name***, L=***locality*, **S=***state***, C=***country_code*

Where:

common_name is the common name of a person or a server host name. Example: **Alice Smith** or **myserver.mydomain.com**.

organization_unit is the department or division of an organization. Example: **Operations**.

organization_name is the organization or company. Example: **Acme**.

locality is the locality such as city. Example: **Redwood City**.

state is the state or province. Example: **California**.

country_code is the two-letter country code per ISO 3166-1. Example: **US**.

If a value contains a comma, it must be escaped with the "\" character. Example: **Acme\, Inc.**

Setting Up the PKI

This section explains how to use the **keytool** program to set up the PKI between administration clients, Processing Servers, and Signaling Servers using self-signed X.500 certificates.

To set up the PKI with self-signed certificates:

1. Generate the public and private keys for each Processing Server and Signaling Server. Repeat this for each server:
   a. Generate the keys for the server:

keytool -genkeypair "*distinguished_name***" -alias** *server_keystore_entry* **-keypass** *key_password* **-keystore** *server_keystore* **-storepass** *server_keystore_password*

Where:

distinguished_name is a X.500 distinguished name for the issuer of the certificate. See "About Keytool and X.500 Distinguished Names".

server_keystore_entry is a keystore entry for the certificate chain and the private key for the server.

key_password is a password used to protect the private key.

server_keystore is a name of the server keystore. Oracle suggests that you name your keystore so it can be identified with the server it will be used with. Example: **ssu_server_1_keystore**, **ssu_server_2_keystore**, and so on.

server_keystore_password is the password that protects the server keystore.

b. Export the public key from the server keystore entry into a self-signed certificate:

keytool -exportcert -alias *server_keystore_entry* **-keystore** *server_keystore* **-storepass** *server_keystore_password* **-rfc -file** *server_certificate***.cer**

Where:

server_keystore_entry is the keystore entry for the certificate chain and the private key for the server. Same as the *server_keystore_entry* used when the keys were generated in the previous step.

server_keystore is the name of the server keystore to export the certificate from. Same as the *server_keystore* used when the keys were generated in the previous step. Example: **ssu_server_1_keystore**, **ssu_server_2_keystore**, and so on.

server_keystore_password is the password setup to protect the server keystore. Same as the *server_keystore_password* used when the keys were generated in the previous step.

server_certificate is the name of the certificate. This certificate shall be imported to each administration client's truststore. Oracle suggests that you name your certificate so it can be identified with the server it originates from. Example: **ssu_server_1.cer**, **ssu_server_2.cer**, and so on.

2. Generate the public and private keys for each administration client. Repeat this for each administration client:

a. **keytool -genkeypair "***distinguished_name***" -alias** *client_keystore_entry* **-keypass** *key_password* **-keystore** *client_keystore* **-storepass** *client_keystore_password*

Where:

distinguished_name is a X.500 distinguished name for the issuer of the certificate.

client_keystore_entry is a keystore entry for the certificate chain and the private key for the administration client.

key_password is a password used to protect the private key.

client_keystore is a name of the administration client keystore. Oracle suggests that you name your keystore so it can be identified with the administration client it will be used with. Example: **client_1_keystore**, **client_2_keystore**, and so on.

client_keystore_password is the password that protects the administration client keystore.

b. Export the public key from the administration client keystore entry into a self-signed certificate:

keytool -exportcert -alias *client_keystore_entry* **-keystore** *client_keystore* **-storepass** *client_keystore_password* **-rfc -file** *client_certificate***.cer**

Where:

client_keystore_entry is the keystore entry for the certificate chain and the private key for the administration client. Same as the *client_keystore_entry* used when the keys were generated in the previous step.

client_keystore is the name of the administration client keystore to export the certificate from.

client_keystore_password is the password setup to protect the administration client keystore. Same as the *client_keystore_password* used when the keys were generated in the previous step.

client_certificate is the name of the certificate. This certificate is imported to each server's truststore. Oracle suggests that you name your certificate so it can be identified with the administration client it originates from. Example: **client_1.cer**, **client_2.cer**, and so on.

3. Import the server certificate into the administration client truststore. Repeat this for each administration client:

keytool -importcert -file *server_certificate* **-keystore** *client_truststore* **-storepass** *client_truststore_password* **-noprompt**

Where:

server_certificate is the name of the server certificate. Example: **ssu_server_1.cer**, **ssu_server_2.cer**, and so on.

client_truststore is a name for the administration client truststore.

client_truststore_password is the password that protects the administration client truststore.

4. Import the administration client certificate into the server truststore. Repeat this for each server:

keytool -importcert -file *client_certificate* **-keystore** *server_truststore* **-storepass** *server_truststore_password* **-noprompt**

Where:

client_certificate is the name of the administration client certificate. Example: **client_1.cer**, **client_2.cer**, and so on.

server_truststore is a name for the server truststore.

server_truststore_password is the password that protects the server truststore.

5. Distribute the keystores and truststores to the servers and administration clients.

Note: The keystores and trustores might be in different directories and might have different names than specified below. The following directories and filenames are according to default settings. See "Defining Keystores and Truststores".

---

a. Identify which keystore-truststore pair belongs to which server or administration client.

b. Copy the keystore-truststore pair to the correct server. Repeat this for each server:

   Copy or use FTP to put the keystore in:

   *Oracle_home***/ocsb60/managed_server/serverkeystore**

   Copy or use FTP to put the truststore in:

   *Oracle_home***/ocsb60/managed_server/servertruststore**

c. Copy the keystore-truststore pair to the correct administration client. Repeat this for each administration client:

   Copy or use FTP to put the keystore in:

   *Oracle_Home***/ocsb60/admin_console/clientkeystore**

   Copy or use FTP to put the truststore in:

   *Oracle_Home***/ocsb60/admin_console/clienttruststore**

Defining the Keystore for Administration Console SSL Connections

The public certificate used to secure the connection between the Web Administration Console and the Web Administration Console server is stored in a keystore accessed by the Web Administration Console server.

The certificate can be stored in either the same keystore as the Web Administration Console server uses when it authenticates with the Processing Servers and the Signaling Servers or it can use a separate keystore.

If the system property **org.eclipse.equinox.http.jetty.ssl.keystore** is not defined, the administration client keystore is used.

If the system property **org.eclipse.equinox.http.jetty.ssl.keystore** is defined, a separate keystore is used.

Using the Administration Client Keystore for Web Administration Console Security

To use the administration client keystore for Web Administration Console security:

1. Make sure the system property **org.eclipse.equinox.http.jetty.ssl.keystore** is not defined. Make sure there is a hash-sign (**#**) before the entry in the file:

   *Oracle_home***/admin_console/properties/web.properties**.

   Example: **#org.eclipse.equinox.http.jetty.ssl.keystore=jettysslkeystore**

2. Open a command shell and change the directory to:

   *Oracle_home***/admin_console**

3. Use Keytool to generate a public and private key for the Web Administration Console server and store them in the administration client keystore:

   keytool -genkeypair "*distinguished_name***" -alias** *keystore_entry* **-keypass** *key_password* **-keystore** *keystore* **-storepass** *keystore_password*

   Where:

   distinguished_name is a X.500 distinguished name for the issuer of the certificate. See "About Keytool and X.500 Distinguished Names".

keystore_entry is a keystore entry for the certificate chain and the private key for the server.

key_password is a password used to protect the private key.

keystore is the name of the keystore defined for the Administration Console keystore. Example: **console_keystore**.

keystore_password is the password that protects the administration client keystore.

4. Use Keytool to export the public key from the keystore entry into a self-signed certificate:

   keytool -exportcert -alias *keystore_entry* **-keystore** *keystore* **-storepass** *keystore_password* **-rfc -file** *certificate***.cer**

   Where:

   keystore_entry is the keystore entry for the certificate chain and the private key for the server. Same as the *keystore_entry* used when the keys were generated in the previous step.

   keystore is the name of the keystore to export the certificate from. Same as the *keystore* used when the keys were generated in the previous step. Example: **console_keystore**.

   keystore_password is the password setup to protect the administration client keystore. Same as the *keystore_password* used when the keys were generated in the previous step.

   certificate is the name of the certificate. This certificate shall be imported to the Web Administration Console server truststore. Oracle suggests that you name your certificate so it can be identified with the Web Administration Console server. Example: **web_console_server.cer**.

5. Use Keytool to import the certificate into the Web Administration Console server truststore:

   keytool -importcert -file *certificate* **-keystore** *truststore* **-storepass** *truststore_password* **-noprompt**

   Where:

   certificate is the name of the server certificate. Example: **web_console_server.cer**.

   truststore is the name for the Web Administration Console server truststore.

   truststore_password is the password that protects the truststore.

Using a Separate Keystore for Web Administration Console Security

To use a separate keystore for Web Administration Console security

1. Make sure the system property **org.eclipse.equinox.http.jetty.ssl.keystore** is defined. The property is defined in the file:

   *Oracle_home***/admin_console/properties/web.properties**

   The value of this property is the filename of and path to the keystore. When using relative paths, it is relative to the directory **admin_console**.

   Example: **org.eclipse.equinox.http.jetty.ssl.keystore=jettysslkeystore**

2. Open a command shell and change the directory to:

   *Oracle_home***/ocsb60/admin_console**.

3. Use Keytool to Generate a public and private key for the Web Administration Console server and store them in the dedicated keystore:

   keytool -genkeypair "*distinguished_name***" -alias** *keystore_entry* **-keypass** *key_password* **-keystore** *keystore* **-storepass** *keystore_password*

   Where:

   distinguished_name is a X.500 distinguished name for the issuer of the certificate. See "About Keytool and X.500 Distinguished Names".

   keystore_entry is a keystore entry for the certificate chain and the private key for the server.

   key_password is a password used to protect the private key.

   keystore is the name of the keystore to use for the connection between the Web Administration Console and the Web Administration Console server. Example: **web_console_keystore**.

   keystore_password is the password that protects the server keystore.

7

Securing Credentials with Credential Store

This chapter explains how to configure and use the Oracle Communications Service Broker (Service Broker) Credential Store feature. You use this feature to securely store, encrypt, and validate the credentials that Service Broker requires to communicate with external non-SSL compliant credential readers.

About Credential Store

Oracle recommends that your Service Broker application-facing network traffic have a minimum security level that is usually satisfied by a combination of firewall protection and HTTPS/SSL/TLS support. Oracle recognizes however, that Service Broker must also communicate with certain non-HTTPS-compliant entities that still require credentials. The Service Broker Credential Store feature is designed to securely store credentials for these types of features.

The Credential Store provides a consistent, efficient, secure way for each OSGI bundle to encrypt, store, and validate credentials for Service Broker features. For example, an SMS-C on a telecom network usually requires a password for access. Credential Store provides a secure place for Service Broker entities such as the SMPP Protocol Adaptor to store and validate that password when accessing SMS-C data and features.

The Credential Store:

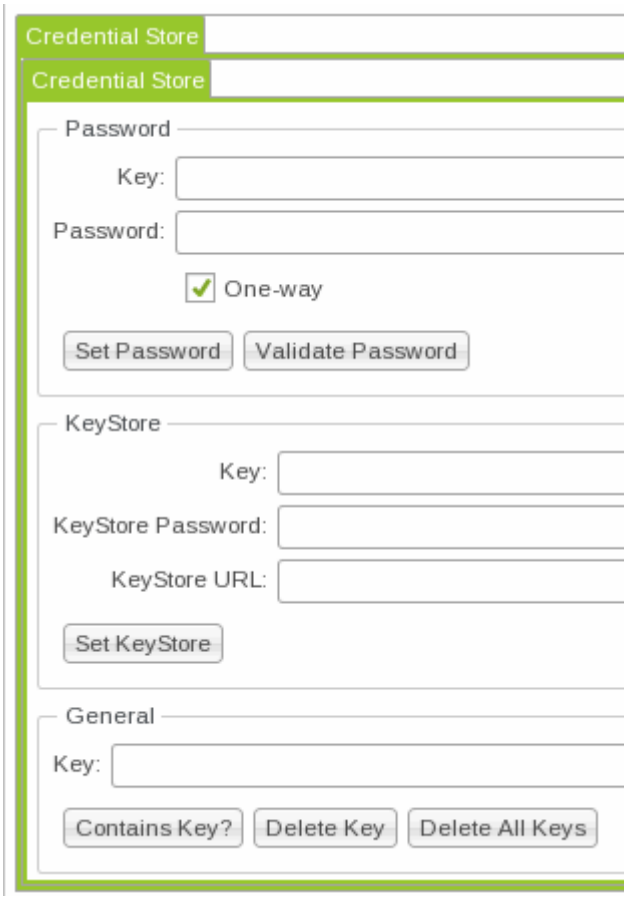
- Stores and validates passwords (1-way encryption).
- Stores and retrieves passwords (2-way encryption)
- Stores and retrieves keystores (databases of credentials).

The Credential Store components include:

- A single encrypted file that contains the credential stores for all OSGi bundles in a domain.
- A master password file to unencrypt each file of credentials.
- GUI interfaces in the Administration Console that you use to add credentials to the credential file singly.
- MBean operations to provision the credential store in bulk

All Service Broker features that use Credential Store have **Credential Store** subtabs in the Administration Console. Figure 7–1 shows the **Credential Store** subtab of the Administration Console **Data Store** tab. The other **Credential Store** subtabs are the same or very similar.

Figure 7–1 Example Credential Store Subtab



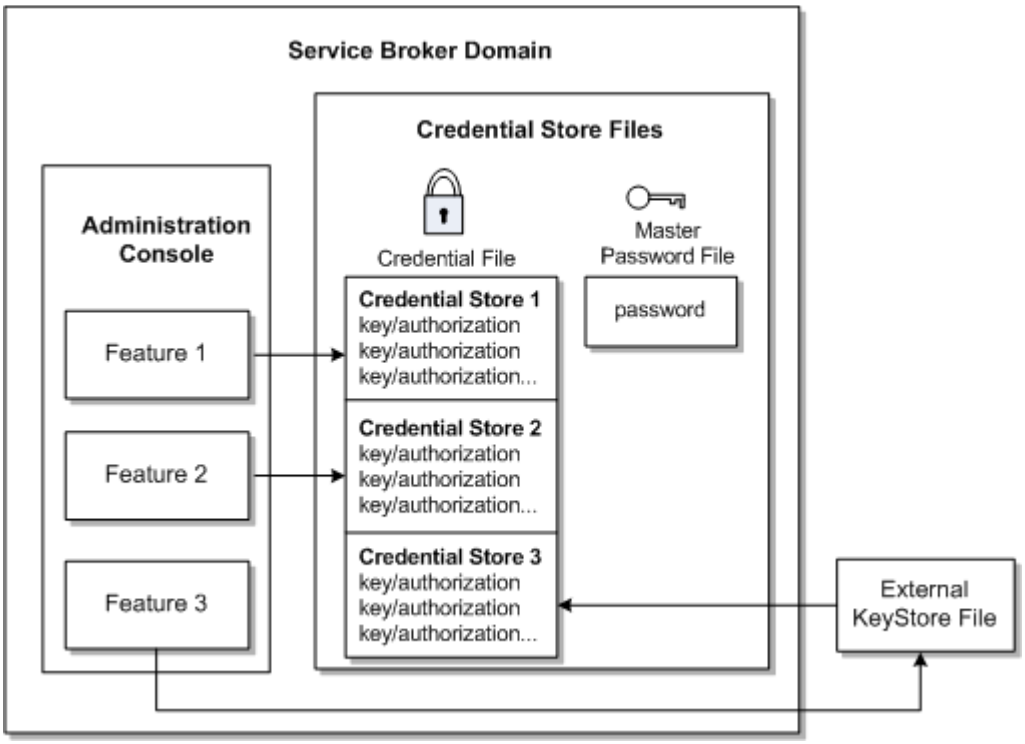
Credential Store works on the Service Broker domain level; all credentials for all OSGi bundles in a domain are stored in a single file. You configure and use Credential Store features through the Administration Console or the MBean interface.

Understanding How Credential Store Works

By default, Service Broker creates a credential store file in each domain for use by the Service Broker components in that domain. Figure 7–2 shows the Credential Store components. Each Service Broker domain contains two files, a *credential file* and a *master password file*. The credential file stores the credentials and the master password file unencrypts the credential file.

Each credential that Service Broker stores is composed of two parts: a *key* (such as a username) that identifies the protected entity and an associated *authorization* (such as a password or keystore certificate) to make it accessible. Both the key (sometimes referred to as a CredentialKey) and the authorization are stored as arbitrary strings. For example, a key could be a simple username, a fully qualified LDAP domain name, or a logical alias to a credential.

Figure 7–2 Provisioning Credential Stores



The credential file is encrypted and can be unencrypted with the *master password* stored in the second credential store file, the master password file. A default master master password is created automatically for each credential store file when its Service Broker domain is created. The default master password is a human-readable random string. Once created, you can change this password as necessary.

Table 7–1 shows the default credential store filenames and locations for each domain.

Table 7–1 Credential Store System Properties

Property	Default Value	Description
axia.credential.store.url	*Domain_home*/**protected/cs_store**	Specifies the URL of the credential file. All Service Broker components (for example all managed servers and consoles) that access a domain share the same credential file. This file must also be secured by the operating system file permission settings.
axia.credential.store.key.url	*Domain_home*/**protected/cs_key**	Specifies the URL of the master password protecting the Credential Store file. All Service Broker components that access a domain share the same master password file. This file must also be secured by the operating system file permission settings, and Oracle recommends that you move this file from its default location as an extra level of security. See "Moving the Credential Store Key Files" for more information.

The Credential Store feature creates one set of these credential file/master password file pairs for each Service Broker domain, for use by all OSGi bundles in that domain. However each credential store within the credential file is isolated, so credentials do

not need to be unique within the domain, just unique within one OSGi bundle. You can store identical credentials in different credential stores in the same credential file.

Together the credential file/master password file pairs protect the credentials in the credential store file.

You can change the Credential Store encryption methods to suit your needs. For more information see "Changing Credential Store Encryption Settings".

You administer credentials singly using the Administration Console or in bulk using an MBean interface. See "Provisioning and Administering Credential Stores" and "Using the Credential Store Management API" for more information.

Considerations for Using Credential Store with Hosted Domains

Using Credential Store with hosted domains requires special considerations. To make credentials more secure, the default Credential Store files are located in a special **/protected** directory in each domain. This location is not accessible to other Managed Servers through HTTP.

These are two options for using Credential Store with hosted domains:

- The most secure strategy is to copy the **cs_store** and **cs_key** files from the *Domain_Home*/**protected** location to an identical location on each of Managed Servers accessing the hosted domain. If this default location does not work for your implementation, you can change it by specifying a different path in the Java system properties using: **axia.credential.store.url** and **axia.credential.store.key.url**. For example:

  ```
  -Daxia.credential.store.url=file:///local_path/cs_store
  -Daxia.credential.store.key.url=file:///local_path/cs_key
  ```

  This approach requires that the **cs_store** file be copied to all Managed Servers every time the Credential Store files change. If the encryption key is ever updated, the **cs_key** file must be copied to each of the domain Managed Servers again, but this is uncommon.

- A “medium” level security alternative is to copy the **cs_key** file from *Domain_Home*/**protected** to an identical location on each of the Managed Servers, and move the **cs_store** file to the Administration Console Domain_Home.

  To make this work, you must change the system properties on both the Administration Console and all of the Managed Servers to specify the new file locations. For example, you could use this system property to start the Administration Console:

  ```
  -Daxia.credential.store.url=file:///Domain_Home/cs_store
  ```

  And use this system property on each of the Managed Servers accessing the hosted domain:

  ```
  -Daxia.credential.store.url=https://Domain_Post_IP:9000/cs_store
  ```

  This approach removes the need to manually copy the **cs_store** file for each Credential Store update, but still protects the encryption key from being shared over the network.If the encryption key is ever updated, the **cs_key** file will need to be copied to each of the domain Managed Servers again, but this is uncommon.

Configuring the Credential Store Domain Settings

The following sections explain the options for changing Credential Store settings. The settings in the following sections apply to *all* features that use Credential Store in a single domain. These settings apply to the entire Credential Store file. Because they apply to all features within a domain using Credential Store, you should only set them once, before you start storing credentials in them.

Moving the Credential Store Key Files

The default **/protected** location that Service Broker uses for the credential store files is a special area of the domain is that is more secure than most. It is accessible by the Administration Console but not the managed server.

You can move the Credential Store files to new locations as desired by renaming the Credential Store path in the System Property.

> **Note:** For extra security, Oracle recommends that you move the **cs_key** master password file to a non-default location and give it the minimum possible access permissions.

Changing Credential Store Encryption Settings

This section explains how to change the Credential Store password storage configuration settings. Table 7–2 lists the default encryption methods and the system properties you use to change them.

> **WARNING: Previously stored credentials become unavailable if you change encryption settings. Make encryption setting changes before you start storing credentials.**

Table 7–2 Credential Store Default Encryption Settings

System Property in common.properties	Default Value	Notes
oracle.axia.credential.store.cipher.algorithm	AES	Sets the cipher algorithm that encrypts/decrypts data.
oracle.axia.credential.store.key.generator.alorithm	AES	Sets the algorithm that generates secret keys.
oracle.axia.credential.store.digest.algorithm	SHA-256	Sets the algorithm that hashes data.

To change these settings add these entries to the *Oracle_home***/ocsb60/admin_console/properties/common.properties** file and then restart the domain.:

```
oracle.axia.credential.store.cipher.algorithm=DES
oracle.axia.credential.store.key.generator.algorithm=RC2
oracle.axia.credential.store.digest.algorithm=
```

Table 7–3 lists the types of algorithms to use. See this discussion on *Java Cryptography Architecture Standard Algorithm Name Documentation* for more information on the encryption algorithms available:
http://docs.oracle.com/javase/6/docs/technotes/guides/security/StandardNames.html.

Table 7–3 Credential Store Algorithm Types

System Property in common.properties	Type of Algorithm
oracle.axia.credential.store.cipher.algorithm	Cipher
oracle.axia.credential.store.key.generator.alorithm	KeyGenerator
oracle.axia.credential.store.digest.algorithm	MessageDigest

Backing Up Credential Store files

Ensure that your domains are backed up, which also backs up the Credential Store files they contain. For more information, see the discussion on backing up your installation in the *Oracle Communications Service Broker System Administrator's Guide*.

Provisioning and Administering Credential Stores

Each Service Broker feature that uses Credential Store has an Administration Console subtab that looks like the one in Figure 7–1. The following sections describe how to use these subtabs to add credentials to the credential store files and administer them. Figure 7–2 illustrates Credential Store provisioning.

These actions operate on one credential at a time. This may be appropriate for test and evaluation systems and small implementations. Service Broker offers an MBean interface to manage credentials in bulk. See "Using the Credential Store Management API" for more information.

Storing or Validating a Credential (Password) in the Credential Store

To add a credential to the Credential Store:

1. Open the Administration Console and navigate to your feature's **Credential Store** tab or subtab.
2. In the **Password** area of the screen enter the following:
   - Enter a credential key (such as a username) in the **Key** field.

     You can store any credential that can be read as a string.
   - Enter a password in the **Password** field.
3. (Optional) Check the **One-Way** box to restrict the credential to read-only use.
4. Do one of the following:
   - Click **Set Password** to add the credential to the credential file.
   - Click **Validate Password** to confirm that this credential already exists in the credential file.

Storing a Keystore in a Credential Store

To add a keystore from a keystore file to a Credential Store:

1. Open the Administration Console and navigate to your feature's **Credential Store** tab or subtab.
2. In the **Keystore** area of the screen, enter the following:
   - The keystore name in the **Key** field.
   - The keystore password in the **KeyStore Password** field.

- The keystore URL in the **KeyStore URL** field. This is the location of the keystore file.

3. Click the **KeyStore** button to add the keystore to the credential file.

Verifying that a Credential Exists in a Credential Store

Use these steps to verify whether a credential exists in a credential store file. These steps to not validate the credential.

1. Open the Administration Console and navigate to your feature's **Credential Store** tab or subtab.
2. In the **General** area of the screen, enter key name in the **Key** field
3. Click the **Contains Key?** button.

Deleting Credentials from the Credential Store

Use the following steps to delete credentials from a Credential Store.

1. Open the Administration Console and navigate to your feature's **Credential Store** tab or subtab.
2. In the **General** area of the screen, enter key name in the **Key** field
3. Click one of the following buttons:
   - Click the Delete Key button to remove the credential from the credential file
   - Click the Delete All Keys button to remove all credentials in the credential file.

Using the Credential Store Management API

The Service Broker Credential Store feature includes the **oracle.axia.api.management.credentialstore.CredentialStore** MBean that you use to to manage credentials for OSGi bundles in bulk. The Administration Console also allows you to manage credentials, but only singly.

The following section lists the Credential Store operations for administration and management of credentials, including:

- setPassword - Adds passwords to a Credential Store.
- validatePasssword - Validates passwords stored in a Credential Store.
- setKeystore - Adds keystore credentials to a Credential Store.
- containsKey - Confirms that a credential exists in a Credential Store
- deleteKey - Deletes keystore credentials from a Credential Store
- clear (Removes all Credentials) - Removes all credentials from a Credential Store.

setPassword

Adds a credential (key and authorization) to the Service Broker bundle's credential store file.

Syntax:

```
void setPassword(@Name("Key") String key, @Name("Password") String password,
@Name("One-way") boolean oneWay)
```

Where:

- **Key** - The credential key.
- **Password** - The authorization string to use for the credential.
- **one-way | two-way** - **one-way** encrypts the password and prevents it from being decrypted later (most secure). **two-way** encrypts the password and allows it to be decrypted later.

validatePasssword

Validates credential information against a credential stored in a credential store file. Returns true if the two are identical.

Syntax:

```
boolean validatePassword(@Name("Key") String key, @Name("Password") String
password)
```

Where:

- **Key** - The credential key.
- **Password** - The authorization string to use for the credential.

setKeystore

Obtains a keystore from a keystore file and stores it in a Credential Store file. Requires a keystore key, password, and the URL of the keystore file.

Syntax:

```
void setKeystore(@Name("Key") String key, @Name("KeyStore Password") String
password, @Name("KeyStore URL") String url)
```

Where:

- **Key** - The credential key for the keystore.
- **KeyStore Password** - The authorization to use for the keystore.
- **KeyStore URL** - The URL location of the keystore file.

containsKey

Returns true if the key exists in the Credential Store; false otherwise.

Syntax:

```
boolean containsKey(@Name("Key") String key)
```

Where:

- **key** - The credential key to search for.

deleteKey

Deletes a credential from a credential store file.

Syntax:

```
void deleteKey(@Name("Key") String key)
```

Where:

- **key** - The credential to delete.

clear (Removes all Credentials)

This operation removes all credentials associated with an OSGi bundle from a credential store file.

Syntax:

```
void clear()
```

8

Managing the Domain Web Server

This chapter describes how to manage the domain web server.

About the Domain Web Server

In hosted domains the domain contains a domain Web servers that provides HTTP access to the domain configuration and the OSGi bundles for the domain. Domain servers access the domain configuration over a web connection.

After creating 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***/ocsb60/admin_console**

   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.

5. If Basic Authentication is enabled, enter the user name and password combination to use for HTTP Basic Authentication.

When the domain web server is started, servers can access the domain configuration over HTTP or HTTPS.

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.

9

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 9–1 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 9–1 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"ServersMBean" and "ServerMBean" for more information.

ServersMBean

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

ServerMBean

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

10

Managing Clusters

This chapter describes how to manage clusters.

About Clustering

Processing Servers and Signaling Servers distribute events among each other across the domain boundaries. To make the domains aware of each other, you need to group them in a cluster.

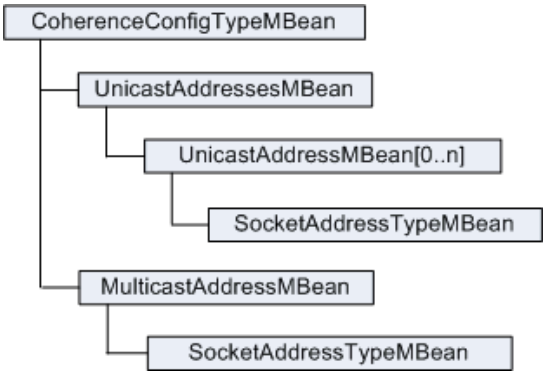
You can group the domains into clusters using one of the following way:

- Using well-known addresses. In this case, servers in one domain are aware of the IP-addresses, or host-names, of all servers in all other domains (see "Grouping Domains Using Well-Known Addresses" for more information).
- Using IP-multicast. This is the default setting when creating the domain using scripts (see "Grouping Domains Using IP-Multicast Addresses" for more information).

These two methods are mutually exclusive. You need to use one or the other on all nodes. Which method to use depends on your network topology, the number of servers, and other considerations.

You can group domains into clusters using CoherenceConfigTypeMBean. Figure 10–1 shows the CoherenceConfigTypeMBean hierarchy. CoherenceConfigTypeMBean contains UnicastAddressesMBean and MulticastAddressMBean.

Figure 10–1 Coherence Configuration MBean Hierarchy



Grouping Domains Using Well-Known Addresses

To group domains using well-known addresses, perform the following steps in all domains:

1. Set the **UseWellKnownAddresses** attribute of **CoherenceConfigTypeMBean** to **true**.
2. Invoke the following operation of **UnicastAddressesMBean**:

   ```
   void addUnicastAddress()
   ```

   A new instance of UnicastAddressMBean that represents the address of a remote server is added to the MBean structure under **UniCastAddressesMBean**.
3. Set the **serverName** attribute of **UnicastAddressMBean** to the name of the server for which you define the IP-unicast address. The name must correlate to the name of the server as specified in the domain configuration.

   Note: The name must be unique across all domains.

4. Set the **address** attribute of **SocketAddressTypeMBean** to the host-name or IP-address that the server uses for clustering.

   The value of the **address** attribute corresponds to the Coherence XML configuration element *address* with the parent-element hierarchy *well-known-addresses* and *socket-address*.
5. Set the attribute **Port** on the MBean **SocketAddressTypeMBean** to the port the server uses for clustering.

   The value of the Port attribute corresponds to the Coherence XML configuration element *port* with the parent-element hierarchy *well-known-addresses* and *socket-address*.

See "CoherenceConfigTypeMBean" for more information about these MBeans.

For example, when servers **pn_1** and **pn_2** are defined in the Processing Domain and the servers **ssu_1** and **ssu_2** are defined in the Signaling Domain, you need to define the following in both the Processing Domain and in the Signaling Domain:

- A **UnicastAddressMBean** with the **serverName** attribute set to **ssu_1**.

  In the sibling **SocketAddressTypeMBean**, you set the **address** attribute to the IP-address the server **ssu_1** will use for clustering and the **Port** attribute to the port the server **ssu_1** will use for clustering.
- A **UnicastAddressMBean** with the **serverName** attribute set to **ssu_2**.

  In the sibling **SocketAddressTypeMBean**, you set the **address** attribute to the IP-address the server **ssu_2** will use for clustering the attribute **Port** to the port the server **ssu_2** will use for clustering.
- A **UnicastAddressMBean** with the **serverName** attribute set to **pn_1**.

  In the sibling **SocketAddressTypeMBean**, you set the **address** attribute to the IP-address the server **pn_1** will use for clustering and the **Port** attribute to the port the server **pn_1** will use for clustering.
- A **UnicastAddressMBean** with the **serverName** attribute set to **pn_2**.

In the sibling **SocketAddressTypeMBean** you set the **address** attribute to the IP-address the server **pn_2** will use for clustering and the **Port** attribute to the port the server **pn_2** will use for clustering.

Grouping Domains Using IP-Multicast Addresses

To group a Processing Domain and a Signaling Domain using IP-multicast addresses, perform the following steps in all domains:

1. Set the **UseWellKnownAddresses** attribute of **CoherenceConfigTypeMBean** to **false**.
2. Set the **address** attribute of **SocketAddressTypeMBean** to the multicast IP address that a socket listens on or publishes to.

   The address attribute corresponds to the Coherence XML configuration element *address* with the parent-element hierarchy *multicast-listener*.
3. Set the **port** attribute of **SocketAddressTypeMBean** to the port that the socket listens or publishes on.

   The port attribute corresponds to the Coherence XML configuration element *port* with the parent-element hierarchy *multicast-listener*.
4. To specify the time-to-live setting for the multicast, set the **Ttl** attribute of **SocketAddressTypeMBean**.

   The Ttl attribute corresponds to the Coherence XML configuration element *ttl* with the parent-element hierarchy *multicast-listener*.

See "CoherenceConfigTypeMBean" for more information about these MBeans.

CoherenceConfigTypeMBean

JAR File

oracle.axia.storage.provider.coherence-*version*.jar

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

Package

oracle.axia.config.beans.storage.coherence

Object Name

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

Number of Allowed Occurrences

Maximum: 1

Factory Method

oracle.axia.config.beans.storage.coherence.ObjectFactory

Attributes

boolean UseWellKnownAddress
Set to:

- **true** to use well-known addresses.
- **false** to use IP-multicast.

See "About Clustering".

Operations

void addMulticastAddress()
Creates a **MulticastAddressMBean**.

MulticastAddressMBean getMulticastAddressMBean()
Gets a **MulticastAddressMBean**.

UnicastAddressesMBean getUnicastAddressesMBean()
Gets a **UnicastAddressesMBean**.

void setMulticastAddressMBean(MulticastAddressMBean val)
Sets a **MulticastAddressMBean**.

Parameter:

val Object of type **MulticastAddressMBean**.

void setUnicastAddressesMBean(UnicastAddressesMBean val)
Sets a **UnicastAddressesMBean**.

Parameter:

val Object of type **UnicastAddressesMBean**.

void removeMulticastAddress()
Removes a **MulticastAddressMBean**.

MulticastAddressMBean

JAR File

oracle.axia.storage.provider.coherence-*version*.jar

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

Package

oracle.axia.config.beans.storage.coherence

Object Name

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

Number of Allowed Occurrences

Maximum: 1

Factory Method

oracle.axia.config.beans.storage.coherence.ObjectFactory

Attributes

int Ttl
Specifies the time-to-live setting for the multicast.

It corresponds to the Coherence XML configuration element *ttl* with the parent-element hierarchy *multicast-listener*. See *Operational Configuration elements* in *Oracle Coherence Developer's Guide Release 3.7*.

Operations

void setSocketAddressMBean(SocketAddressTypeMBean val)
Sets a **SocketAddressTypeMBean**.

Parameter:

val Object of type **SocketAddressTypeMBean**.

SocketAddressTypeMBean getSocketAddressMBean()
Gets a **SocketAddressTypeMBean**.

SocketAddressTypeMBean

JAR File

oracle.axia.storage.provider.coherence-*version*.jar

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

Package

oracle.axia.config.beans.storage.coherence

Object Name

Depending on the parent MBean, the object name is:

oracle:name=oracle.axia.storage.provider.coherence,name0=coherenceConfiguration,name1=unicastAddresses,name2=unicastAddress[*n*],name3=socketAddress,type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0

or:

oracle:name=oracle.axia.storage.provider.coherence,name0=coherenceConfiguration,name1=multicastAddress,name2=socketAddress,type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0

n is an index for the MBean.

Number of Allowed Occurrences

Maximum: 1

Factory Method

oracle.axia.config.beans.storage.coherence.ObjectFactory

Attributes

String Address

Host-name or IP-address the server uses for clustering.

Depending on which the parent MBean is:

- It corresponds to the Coherence XML configuration element *address* with the parent-element hierarchy *well-known-addresses* and *socket-address*. For details about this setting, see *Operational Configuration elements* in *Oracle Coherence Developer's Guide Release 3.7*.
- It corresponds to the Coherence XML configuration element *address* with the parent-element hierarchy *multicast-listener*.

int Port

IP-port the server the server uses for clustering.

Depending on which the parent MBean is:

- It corresponds to the Coherence XML configuration element *port* with the parent-element hierarchy *well-known-addresses* and *socket-address*.
- It corresponds to the Coherence XML configuration element *port* with the parent-element hierarchy *multicast-listener*.

For details about this setting, see *Operational Configuration elements* in *Oracle Coherence Developer's Guide Release 3.7*

Operations

None

UnicastAddressesMBean

JAR File

oracle.axia.storage.provider.coherence-*version*.jar

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

Package

oracle.axia.config.beans.storage.coherence

Object Name

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

Number of Allowed Occurrences

Maximum: 1

Factory Method

oracle.axia.config.beans.storage.coherence.ObjectFactory

Attributes

None

Operations

void addUnicastAddressMBean()
Creates a **UnicastAddressMBean** MBean. An index is generated for the MBean.

List<UnicastAddressMBean> getUnicastAddressMBean()
Lists the sibling MBeans of type **UnicastAddressMBean**.

void removeUnicastAddress(int index)
Removes the MBean **UnicastAddressMBean**.

Parameter:

index Index of the MBean.

UnicastAddressMBean

JAR File

oracle.axia.storage.provider.coherence-*version*.jar

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

Package

oracle.axia.config.beans.storage.coherence

Object Name

oracle:name=oracle.axia.storage.provider.coherence,name0=coherenceConfiguration,name1=unicastAddresses,name2=unicastAddress[*n*],type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0

n is an index for the MBean.

Number of Allowed Occurrences

Minimum: 0; No maximum

Factory Method

oracle.axia.config.beans.storage.coherence.ObjectFactory

Attributes

String ServerName
The name of the server you are defining the IP-unicast address for. The name must correlate to the name of the server as given in the domain configuration.

Operations

SocketAddressTypeMBean getSocketAddressMBean()
Gets a **SocketAddressTypeMBean**.

void setSocketAddressMBean(SocketAddressTypeMBean val)
Creates a **SocketAddressTypeMBean**.

Parameter:

val Object of type **SocketAddressTypeMBean**.

11

Starting and Stopping the Administration Console

This chapter describes how to configure Service Broker using the Standalone and Web Administration Consoles.

Starting and Stopping the Standalone Administration Console

The Standalone Administration Console is a desktop software application provided with Service Broker.

Starting the Standalone Administration Console

To start the Standalone Administration Console:

1. Log in to the physical system where the Standalone Administration Console 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***/ocsb60/admin_console**

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

3. Enter:

   ./start.sh *Domain_configuration_directory*

   Domain_configuration_directory is the path to the domain configuration directory.

Stopping the Standalone Administration Console

To stop the Standalone Administration Console, use the Administration Console itself. See *Oracle Communications Service Broker Configuration Guide*.

Starting and Stopping the Web Administration Console

You can access the Web Administration Console using a standard web browser.

Starting the Web Administration Console Server

To start a Web Administration Console 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***/ocsb60/admin_console**

   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 to use during the Web Administration Console login procedure.

Logging In to the Web Administration Console

To log in to the Web 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 "Enabling and Disabling SSL" in "Configuring Security Between Service Broker Components" for details.
   - *host* is the IP-address or host name.
   - *port* is the port for the Web Administration Console server. The default value for the port is 9000.

3. If it is the first time you are logging in to the Web 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 Web Administration Console server was started.

Stopping the Web Administration Console Server

To stop the Web Administration Console 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 Console 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 Web Administration Console Server Security

The Web Administration Console server authenticates with the Processing Servers and the Signaling Servers. It also authenticates with the browser that displays the Web Administration Console.

For information about setting up security for the Web Administration Console Server, see "Configuring Security Between Service Broker Components"

12

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 with 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 have to 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 12–1.
3. Click **Apply**.

Removing a Server from a Domain

Before you remove a server from the Domain Configuration you have to 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.

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 12–1 parameters that you need to provide.

Table 12–1 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.

See "DomainServiceMBean" for more information about this MBean.

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 "Grouping Domains Using Well-Known Addresses" for more information.

3. Configure security for the server. See "Configuring Security Between Service Broker Components" 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)
   ```

   See "DomainServiceMBean" for more information about this MBean.

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.

13

Starting and Stopping Processing Servers and Signaling Servers

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

Starting a Processing Server or a Signaling Server

After you added a server to a domain, you can start this 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***/ocsb60/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.

5. If you are using Basic authentication you are prompted for the user name and password combination. Use the same user name and password you used when you started the domain.

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 13–1 shows a sample code that stops a server.

Example 13–1 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 Scripts for Configuration and Management" for more information on the Scripting Engine.

14

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 receive 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***/ocsb60/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 relevant commands of the operating system (for example, **kill**).

15

Upgrading and Patching Service Broker

This chapter describes how you upgrade your installation and how to apply patches. It also describes the DeploymentServiceMBean.

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.

Overview of Performing an In-Production Upgrade

An in-production upgrade means that Oracle Communications Service Broker is upgraded while continuing to process requests. Servers are upgraded one at a time, in sequential order, until all servers have been upgraded. Service Broker continues to process requests during the upgrade.

During the upgrade procedure, all servers except the server currently being upgraded continue to process traffic. During the upgrade procedure, one set of servers is still on pre-upgrade level, while another set is on post-upgrade level.

To perform an in-production upgrade:

1. Back up your Processing Servers, Signaling Servers, and domain configurations prior to performing an upgrade. See "Maintaining Oracle Communications Service Broker" for details.
2. Do one of the following according to whether the patch is for a bundle or for a server:
   - Patch is for a bundle: Apply the patch to the domain configuration. See "Applying Patches for Bundles" for details.
   - Patch is for a server: Apply the patch to each server. See "Applying Patches for Servers" for details.
3. On each server:

   Perform a controlled shutdown of the server and start it again. See "Starting and Stopping Processing Servers and Signaling Servers".

Applying Patches for Bundles

To install a patch or a patch set for a bundle:

1. Log in to the computer where the Administration Console is installed.
2. Unzip the ZIP file to the directory *patch_directory*.

   patch_directory can be any directory.
3. Do one of the following according to whether the patch or patch set is bundled with Oracle Universal Installer:

   Bundled with Oracle Universal Installer:

   1. In a command shell, navigate to the installer directory:

      patch_directory`/Disk1/install`
   2. Enter the following command to launch the installer:

      `./runInstaller.sh`

      The Oracle Universal Installer is launched in graphical mode.

   Not bundled with Oracle Universal Installer:

   1. Start a local copy of Oracle Universal Installer.
   2. In Oracle Universal Installer, open the file

      patch_directory`/Disk1/install/product.xml`
4. Follow the instructions in the installer. Use the same *Oracle_home* you used when you installed Oracle Communications Service Broker.

   When the installer is finished, the patch bundle(s) are installed in the directory:

   Oracle_home/**patch**
5. Set the domain configuration to offline by setting the attribute **OffLine** on the **DomainServiceMBean** to **true**. See the discussion on creating the domain in *Service Broker Installation Guide* for details.
6. Uninstall the bundle(s) you want to update using the operation **uninstallBundle** in **DeploymentServiceMBean** method. See "DeploymentServiceMBean".
7. Install the updated bundle(s) using the operation **installBundle** in **DeploymentServiceMBean**. See "DeploymentServiceMBean".

Now the domain configuration is updated and the next step is to apply the updated configuration to the servers in the domain by restarting them one by one.

Applying Patches for Servers

To install a patch or a patch set for a server:

1. Log in to the computer where the server to be patched is installed.
2. Unzip the ZIP file to the directory *patch_directory*.

   patch_directory can be any directory.
3. If the patch or patch set is bundled with Oracle Universal Installer:

   1. In a command shell, navigate to the installer directory:

      patch_directory`/Disk1/install`

2. Enter the following command to launch the installer:

   `./runInstaller.sh`

   The Oracle Universal Installer is launched in graphical mode.

4. If the patch or patch set is not bundled with Oracle Universal Installer:
   1. Start a local copy of Oracle Universal Installer.
   2. In Oracle Universal Installer, open the file

      patch_directory`/Disk1/install/product.xml`
5. Follow the instructions in the installer. Use the same *Oracle_home* you used when you installed Oracle Communications Service Broker.

When the installer is finished, the server is updated with the patch or patch set. The server needs to be restarted.

Managing Domain Bundles

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

Table 15–1 describes OSGi Bundle properties.

Table 15–1 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 15–1.

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 have to 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. See "DeploymentServiceMBean" for details on using this MBean.

DeploymentServiceMBean

The DeploymentServiceMBean provides operations for OSGi bundle management.

JAR File

oracle.axia.deployment.adminservice.api-*version*.jar

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

Package

oracle.axia.api.management.deployment

Object Name

oracle:type=oracle.axia.api.management.deployment.DeploymentServiceMBean

Factory Method

Created automatically.

Attributes

None

Operations

String[] deployBundle(String bundleLocation, int startLevel)
Installs and starts an OSGi bundle.

Parameters:

- **bundleLocation** Path to the bundle to deploy.
- **startLevel** OSGi start level of the bundle.

String[] installBundle(String bundleLocation, int startLevel)
Installs an OSGi bundle.

Parameters:

- **bundleLocation** Path to the bundle to deploy.
- **startLevel** OSGi start level of the bundle.

List<Map<String,Serializable>>listDeployedBundles()
List a all deployed OSGi bundles.

void startBundle(String bundleName, String bundleVersion)
Starts an OSGi bundle.

Parameters:

- **bundleName** Symbolic name of the OSGi bundle.
- **bundleVersion** Version of the bundle.

void stopBundle(String bundleName, String bundleVersion)
Stops an OSGi bundle.

Parameters:

- **bundleName** Symbolic name of the OSGi bundle.
- **bundleVersion** Version of the bundle.

void undeployBundle(String bundleName, String bundleVersion)
Undeploys an OSGi bundle.

Parameters:

- **bundleName** Symbolic name of the OSGi bundle.
- **bundleVersion** Version of the bundle.

void uninstallBundle(String bundleName, String bundleVersion)
Uninstalls an OSGi bundle.

Parameters:

- **bundleName** Symbolic name of the OSGi bundle.
- **bundleVersion** Version of the bundle.

void updateBundle(String bundleName, String bundleVersion, String bundleLocation)
Updates an installed OSGi bundle with a bundle stored on the file system.

Parameters:

- **bundleName** Symbolic name of the OSGi bundle.
- **bundleVersion** Version of the bundle.
- **bundleLocation** Path to the update OSGi bundle.

16

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 Signaling Server

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

Oracle_home/**ocsb60/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 Standalone Administration Console
- The Web Administration Console Server
- The Scripting Engine

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

Oracle_home/**ocsb60/admin_console**

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***/ocsb60/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***/ocsb60/admin_console**

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_console**

These directories are located under:

Linux and Solaris: *Oracle_home*/**ocsb60**

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/

17

Life Cycle of Processing Servers and Signaling Servers

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

Life Cycle

Oracle Communications Service Broker has life cycle states and transitions, as illustrated in Figure 17–1. 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 17–1 shows Service Broker states and transitions.

Figure 17–1 States and State Transitions

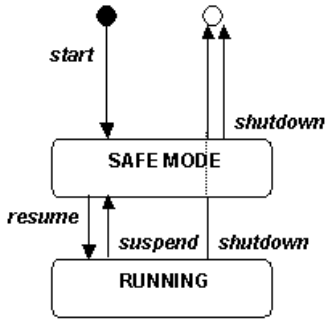


Table 17–1 describes the OSGi states and state transitions, and gives information on the corresponding OSGi start levels.

Table 17–1 OSGi Start Levels, Service Broker States, and State Transitions

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

Table 17–1 (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		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		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 "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.

18

Monitoring Service Broker

This chapter explains how to monitor Oracle Communications Service Broker.

Introduction to Service Broker Monitoring

Service Broker monitoring is based on JMX and you can monitor Service Broker using JMX Runtime MBeans. JMX Runtime MBeans 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.

Understanding Service Broker Runtime MBeans

This section describes the Service Broker Runtime MBeans.

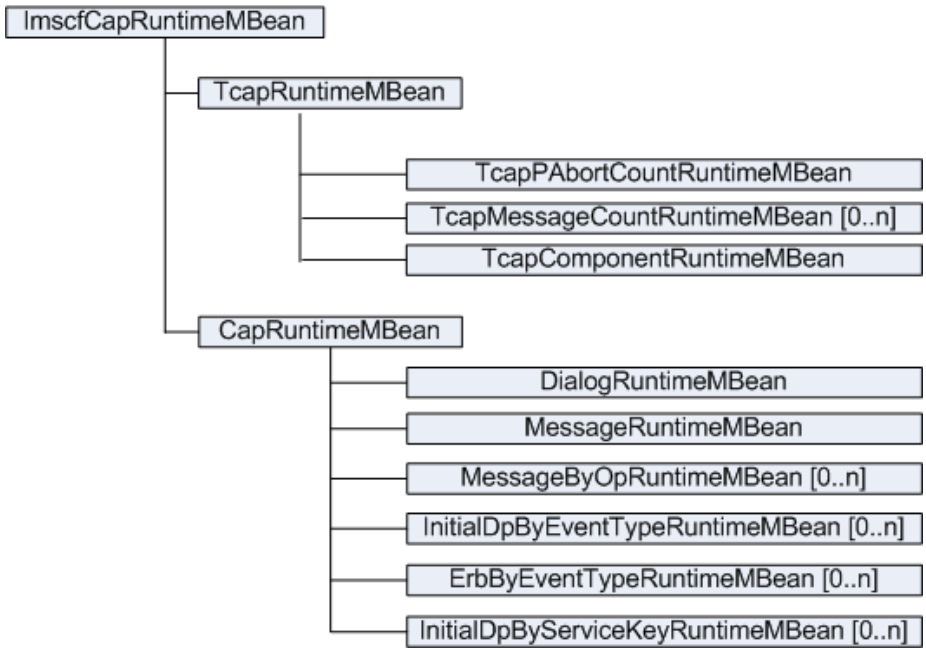
Service Broker Runtime MBeans Organization

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 18–1 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 18–1 Example of the IM-SCF CAP Runtime MBean Hierarchy



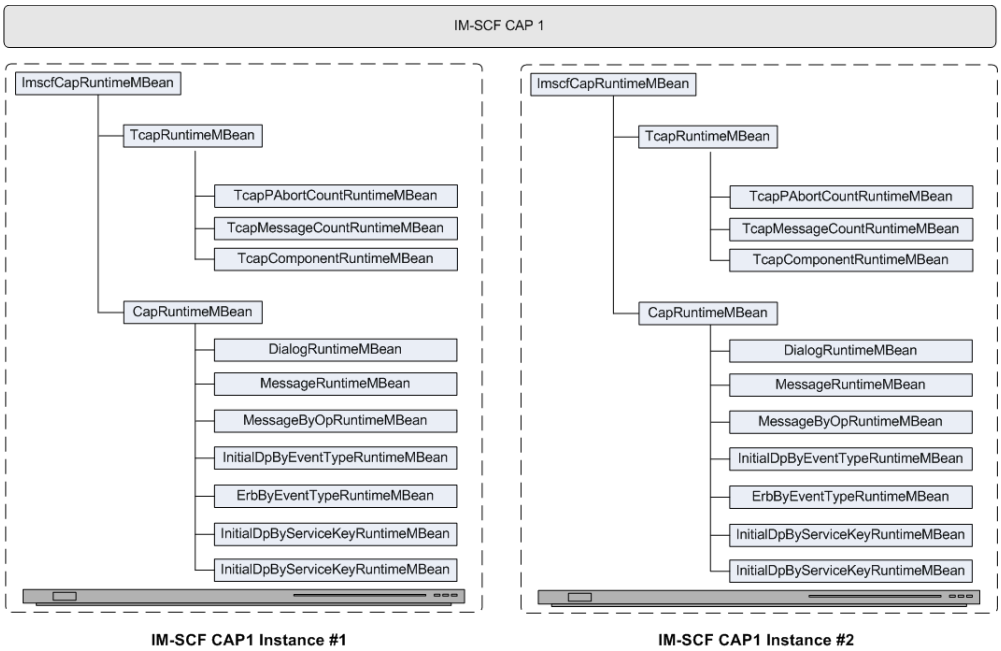
Each Runtime MBean provides reference to other Runtime MBeans under it in the hierarchy. Therefore, a JMX client can access a Runtime MBean by either directly looking it up in the MBean Server or by browsing down through the Runtime MBean hierarchy.

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 18–2 shows an example of how IM-SCF instantiates Runtime MBeans on each of the servers on which IM-SCF is installed.

Figure 18–2 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 18–1 describes the key properties that Service Broker encodes in its Runtime MBean object names.

Table 18–1 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.
Version=*version_number*	Specifies a version of the MBean instance. 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.

Table 18–1 (Cont.) Service Broker MBean Object Name Key Properties

Property	Description
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 till 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. See *Oracle Communications Service Broker Modules Runtime MBeans Javadoc*.

Service Broker Measurements

This section describe various aspects of Service Broker measurements.

Counters, Gauges, TPSs and Statuses

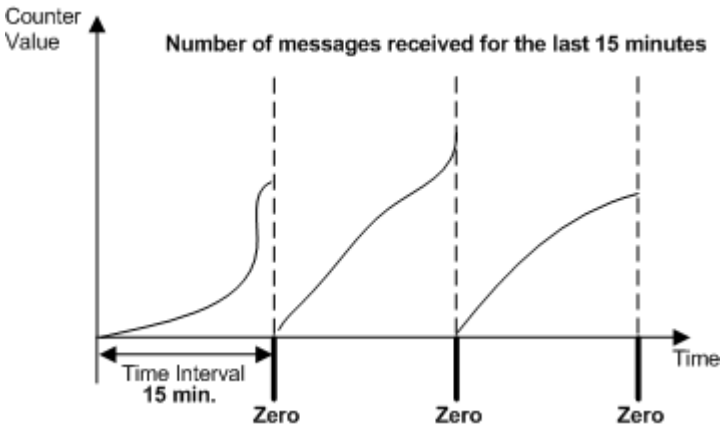
Service Broker Runtime MBeans provide the following types of attributes:

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 configurable for each module instance depending on your specific needs. Whenever a time interval ends, Service Broker zeroes a counter.

Figure 18–3 shows how a counter increases and zeroes periodically in the end of each time interval.

Figure 18–3 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 counters provides the number of times that a specific event occurred from the beginning of the time interval till 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 to act as a current interval counter, create an instance of an MBean and set the CountingMethod property of the MBean object name to CurrentInterval.

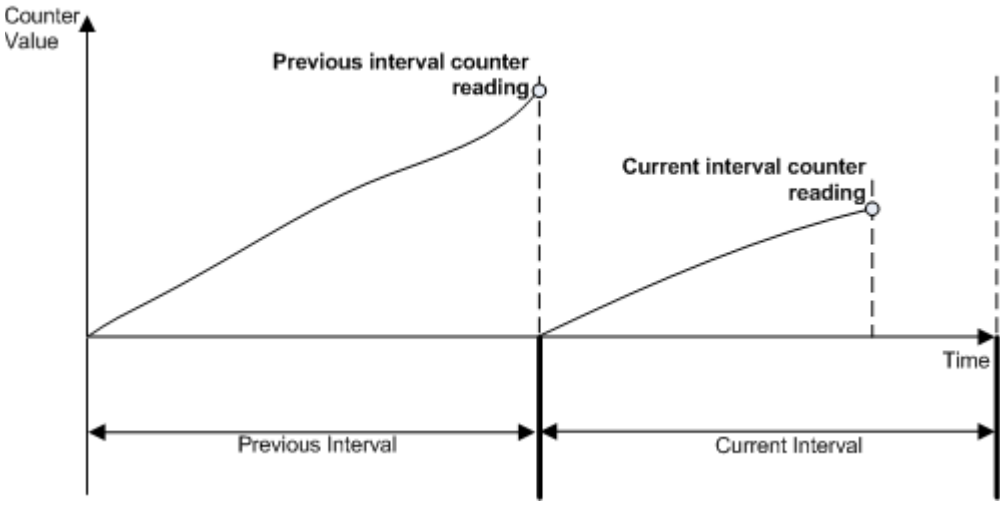
- Previous interval counters

  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 to 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 18–4 shows an example of a current interval counter compared to a previous interval counter.

Figure 18–4 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.

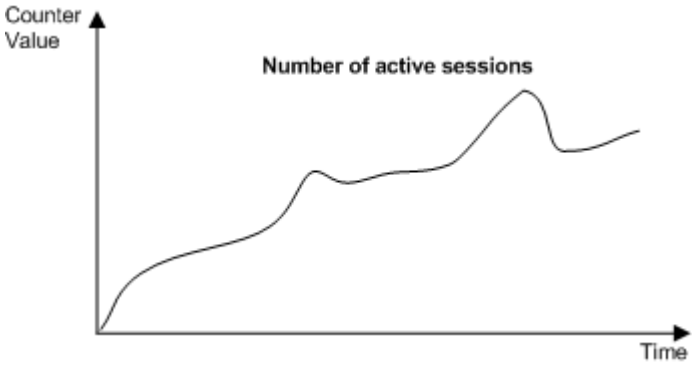
Names of TPS attributes have the **count** prefix. However, as opposed to regular counter attributes, TPS counters are available in MBean instances whose CountingMethod property is set to ShortInterval.

Gauges

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

Figure 18–5 shows how a gauge's value changes over time.

Figure 18–5 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.

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 18–2 explains how reaching an upper threshold or a lower threshold causes reporting and clearing of a notification.

Table 18–2 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 18–3 explains how reaching an upper threshold or a lower threshold causes reporting and clearing of a notification.

Table 18–3 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 18–4 explains how entering a specified state and exiting a state causes reporting and clearing of a notification.

Table 18–4 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

  - *class* defines a type of the change that triggers the notification. Table 18–5 describes notification classes that can be defined in the Type field.

Table 18–5 Notification Class Values

Class	Description
High	A runtime MBean counter or gauge crosses an upper threshold
Low	A runtime MBean counter or gauge crosses a lower threshold
Attribute.changed	A value of a runtime MBean attribute changed
Ceased	A previous notification is cleared

- 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. If you want 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 have to identify the measurements that best serve the evaluation of your system. For the

summary of available Service Broker measurements, see "Summary of Service Broker Measurements".

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

For detailed description on how you can configure the monitoring functionality in the module and tier level, see Configuring Service Broker Monitoring in *Oracle Communications Service Broker Configuration Guide*.

Summary of Service Broker Measurements

This section summarizes measurements that you can take to monitor activity of Service Broker and individual Service Broker's modules.

Monitoring Common Interfaces

The following sections provide reference information on attributes of Runtime MBeans that enable you to monitor interfaces which are common for IMs.

Monitoring the TCAP Interface

Table 18–6 describes attributes with which you can monitor the TCAP interface.

Table 18–6 TCAP Interface Monitoring Attributes

Attribute	Description	Type	MBean
AppInitiatedTransCount	Returns the number of currently opened applications that initiated TCAP Transactions	Counter	TcapRuntimeMBean
DestroyTransactionCount	Returns the number of TCAP transactions that have been closed on the module in the last measurement period	Counter	TcapRuntimeMBean
NetworkInitiatedTrans Count	Returns the number of currently opened TCAP transactions initiated by the network	Counter	TcapRuntimeMBean
TransactionCount	Returns the total number of TCAP transactions that a module handled in the last measurement period	Counter	TcapRuntimeMBean
AbnormalDialogueCount	Returns the number of P-ABORT messages received from the network with the Reason header set to No Common Dialogue Portion	Counter	TcapPAbortCountRuntimeMBean

Table 18–6 (Cont.) TCAP Interface Monitoring Attributes

Attribute	Description	Type	MBean
BadlyFormattedTransactionPortionCount		Counter	TcapPAbortCountRuntimeMBean
IncorrectTransactionPortionCount	Returns the number of TC-P-ABORT messages that a module received received from a network with the Reason header set to Incorrect Transaction Portion in the last measurement period	Counter	TcapPAbortCountRuntimeMBean
NoCommonDialoguePortionCount	Returns the number of the number of TC-P-ABORT messages that a module received from a network with the Reason header set to No Common Dialogue Portion in the last measurement period	Counter	TcapPAbortCountRuntimeMBean
ResourceLimitationCount	Returns the number of TC-P-ABORT messages that a module received received from a network with the Reason header set to Resource Limitation in the last measurement period	Counter	TcapPAbortCountRuntimeMBean
UnrecognizedMessageTypeCount	Returns the number of TC-P-ABORT messages that a module received from a network with the Reason header set to Unrecognized Message Type in the last measurement period	Counter	TcapPAbortCountRuntimeMBean
UnrecognizedTransactionIDCount	Returns the number of TC-P-ABORT messages that a module received received from a network with the Reason header set to'Unrecognized Transaction ID in the last measurement period	Counter	TcapPAbortCountRuntimeMBean
ReceiveFailedCount	Returns the number of messages that a module received but failed to process successfully in the last measurement period	Counter	TcapMessageCount RuntimeMBean
ReceiveSuccessCount	Returns the number of messages that a module received and processed successfully in the last measurement period	Counter	TcapMessageCount RuntimeMBean
SentFailedCount	Returns the number of messages that a module failed to send to the SS7 network in the last measurement period	Counter	TcapMessageCount RuntimeMBean

Table 18–6 (Cont.) TCAP Interface Monitoring Attributes

Attribute	Description	Type	MBean
SentSuccessCount	Returns the number of messages that a module successfully sent to the SS7 network in the last measurement period.	Counter	TcapMessageCount RuntimeMBean
ComponentInvokeTimer Expiry Count	Returns the Invoke component timer expiry count	Counter	TcapComponentRuntime MBean
ComponentRejectTimer ExpiryCount	Returns the Reject component timer expiry count	Counter	TcapComponentRuntime MBean
ReceivedErrorCount	Returns the number of TCAP Error components that a module received from the network in the last measurement period	Counter	TcapComponentRuntime MBean
ReceivedInvokeCount	Returns the number of TCAP Invoke components that a module received from the network in the last measurement period	Counter	TcapComponentRuntime MBean
ReceivedResultLCount	Returns the number of TCAP ResultL components that a module received from the network in the last measurement period	Counter	TcapComponentRuntime MBean
ReceivedResultNLCount	Returns the number of TCAP ResultNL components that a module received from the network in the last measurement period	Counter	TcapComponentRuntime MBean
ReceivedTimerResetCount	Returns the number of received TCAP TimerReset components that a module received from the network in the last measurement period	Counter	TcapComponentRuntime MBean
ReceivedUCancelCount	Returns the number of TCAP UCancel components that a module received from the network in the last measurement period	Counter	TcapComponentRuntime MBean
ReceivedURejectCount	Returns the number of U-REJECT components that a module received from the network in the last measurement period	Counter	TcapComponentRuntime MBean
SentErrorCount	Returns the number of TCAP Error components that a module sent to the network in the last measurement period	Counter	TcapComponentRuntime MBean

Table 18–6 (Cont.) TCAP Interface Monitoring Attributes

Attribute	Description	Type	MBean
SentInvokeCount	Returns the number of TCAP Invoke components that a module sent to the network in the last measurement period	Counter	TcapComponentRuntime MBean
SentLRejectCount	Returns the number of TCAP LReject components that a module sent to the network in the last measurement period	Counter	TcapComponentRuntime MBean
SentResultNLCount	Returns the number of TCAP ResultNL components that a module sent to the network in the last measurement period	Counter	TcapComponentRuntime MBean
SentRRejectCount	Returns the number of TCAP RReject components that a module sent to the network in the last measurement period	Counter	TcapComponentRuntime MBean

Monitoring the IN Interface

Table 18–7 describes attributes that enable you to monitor the IN interface.

Table 18–7 IN INterface Monitoring Attributes

Attribute	Description	Type	MBean
NetworkInitiatedDialog Count	Returns the number of non-module-initiated dialogs that a module handled in the last measurement period	Counter	DialogRuntimeMBean
NetworkInitiatedDialog Gauge	Returns the number of currently opened dialogs initiated by the network	Counter	NOT IMPLEMENTED
ServiceInitiatedDialog Count	Returns the number of dialogs that a module initiated and handled in the last measurement period	Counter	DialogRuntimeMBean
ServiceInitiatedDialog Gauge	Returns the number of currently opened dialogs initiated by a service	Counter	NOT IMPLEMENTED
RcvCount	Returns the number of messages that a module received from the network in the last measurement period	Counter	MessageRuntimeMBean

Table 18–7 (Cont.) IN INterface Monitoring Attributes

Attribute	Description	Type	MBean
SndCount	Returns the number of messages that a module sent to the network in the last measurement period	Counter	MessageRuntimeMBean

Monitoring the Diameter Interface

Table 18–8 describes attributes that enable you to monitor the Diameter interface.

Table 18–8 Diameter Monitoring Attributes

Attribute	Description	Type	MBean
CcaCount	Returns the total number of CCAs that a module sent and received	Counter	RoRuntimeMBean
CcrCalledInitiatorCount	Returns the number of Credit-Control-Requests (CCRs) received with the Media-Initiator-Flag AVP set to CalledParty	Counter	RoRuntimeMBean
CcrCallingInitiatorCount	Returns the number of Credit-Control-Requests (CCRs) received with the Media-Initiator-Flag AVP set to CallingParty	Counter	RoRuntimeMBean
CcrCount	Returns the total number of Credit-Control-Requests (CCRs) that a module sent and received in the last measurement period	Counter	RoRuntimeMBean
CcrUnknownInitiator Count	Returns the number of Credit-Control-Requests (CCRs) received with the Media-Initiator-Flag AVP set to Unknown	Counter	RoRuntimeMBean
EcurSessionCount	Returns the number of Event Charging with Unit Reservation (ECUR) sessions that a module handled in the last measurement period	Counter	RoRuntimeMBean
ErrorCcaCount	Returns the total number of CCAs with an error result that a module sent and received	Counter	RoRuntimeMBean
IecSessionCount	Returns the number of Immediate Event Charging (IEC) sessions that a module handled in the last measurement period	Counter	RoRuntimeMBean

Table 18–8 (Cont.) Diameter Monitoring Attributes

Attribute	Description	Type	MBean
ScurInitialCcrCount	Returns the number of Initial Credit Control Requests (CCRs) that a module sent and received in the last measurement period	Counter	RoRuntimeMBean
ScurSessionCount	Returns the number of Session Charging with Unit Reservation (SCUR) sessions that a module handled in the last measurement period	Counter	RoRuntimeMBean
ScurTerminateCcrCount	Returns the number of Terminate Credit Control Requests (CCRs) that a module sent and received in the last measurement period	Counter	RoRuntimeMBean
ScurUpdateCcrCount	Returns the number of Update Credit Control Requests (CCRs) that a module sent and received in the last measurement period	Counter	RoRuntimeMBean
SuccessCcaCount	Returns the total number of CCAs with a successful result that a module sent and received	Counter	RoRuntimeMBean
AnsCount	Returns the number of Diameter answers that a module sent and received	Counter	DiameterRuntimeMBean
RequestCount	Returns the number of Diameter requests that a module sent and received	Counter	DiameterRuntimeMBean

Monitoring the SIP Interface

Table 18–9 describes attributes that enable you to monitor the SIP interface.

Table 18–9 SIP Interface Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the total number of SIP sessions handled in the last measurement period	Counter	SipRuntimeMBean
SessionGauge	Returns the number of SIP sessions that a module is currently handling	Gauge	NOT IMPLEMENTED

Table 18–9 (Cont.) SIP Interface Monitoring Attributes

Attribute	Description	Type	MBean
RcvCount	Returns the total number of SIP requests that a module received from the network	Counter	SipRequestRuntime MBean SipRequestByMethod RuntimeMBean SipResponseByRequest MethodRuntimeMBean
SndCount	Returns the total number of SIP requests that a module sent to the network	Counter	SipRequestRuntime MBean SipRequestByMethod RuntimeMBean SipResponseByRequest MethodRuntimeMBean

Monitoring Service Broker's Modules

The following sections provide reference information on attributes of Runtime MBeans that enable you to monitor Service Broker's individual modules.

Monitoring the Processing Tier

Table 18–10 describes attributes that enable you to monitor the Service Broker Processing Tier.

Table 18–10 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

Monitoring SS7 SSU

Table 18–11 describes attributes that enable you to monitor SS7 SSU.

Table 18–11 SS7 SSU Monitoring Attributes

Attribute	Description	Type	MBean
Status	Specifies the point code status. Possible values: ■ 1 - Inaccessible ■ 2 - Congested ■ 3 - Accessible	Status	SsuLocalPointCode RuntimeMBean
InService	Returns a subsystem status	Status	SsuLocalSubSystem RuntimeMBean

Table 18–11 (Cont.) SS7 SSU Monitoring Attributes

Attribute	Description	Type	MBean
Status	Specifies a point code status. Possible values: ■ 0 - Remote SCCP is unavailable ■ 1 - Remote SCCP is available	Status	SsuRemotePointCode RuntimeMBean

Monitoring SIP SSU

Table 18–12 describes attributes that enable you to monitor the SIP SSU.

Table 18–12 SIP SSU Monitoring Attributes

Attribute	Description	Type	MBean
Status	Specifies a network entity status: ■ 0 - Network entity is unavailable ■ 1 - Network entity is available ■ 2 - Status of the network entity is unknown	Status	NetworkEntityRuntime MBean
Value	Specifies the monitored network entity. The attribute contains the address of the network entity in the URI format where with the colon character (:) is replaced with the underscore.	Status	NetworkEntityRuntime MBean

Monitoring PCP SSU

Table 18–13 describes attributes that enable you to monitor the PCP SSU.

Table 18–13 PCP SSU Monitoring Attribute

Attribute	Description	Type	MBean
PcpPaStatus	Specifies the status of a BRM applicatikon. Possible values: ■ 0 - Inactive ■ 1 - Active	Status	PcpPaStatisticsMBean

Monitoring SMPP SSU

Table 18–14 describes attributes that enable you to monitor the SMPP SSU.

Table 18–14 SMPP SSU Monitoring Attributes

Attribute	Description	Type	MBean
listSmscConnectionStatus[]	Specifies the status of all SMSC connections. Possible values: ■ 0 - Inactive ■ 1 - Active	Status	SmppAdapterMBean
Status	Specifies a network entity status: ■ 0 - Network entity is unavailable ■ 1 - Network entity is available ■ 2 - Status of the network entity is unknown	Status	NetworkEntityRuntime MBean
Value	Specifies the monitored network entity. The attribute contains the address of the network entity in the URI format where with the colon character (:) is replaced with the underscore.	Status	NetworkEntityRuntime MBean

Monitoring the Diameter SSU

Table 18–15 describes the attributes that enable you to monitor the Diameter SSU.

Table 18–15 Diameter SSU Monitoring Attributes

Attribute	Description	Type	MBean
Status	Specifies a network entity status: ■ 0 - Network entity is unavailable ■ 1 - Network entity is available ■ 2 - Status of the network entity is unknown	Status	NetworkEntityRuntime MBean
Value	Specifies the monitored network entity. The attribute contains the address of the network entity in the URI format where with the colon character (:) is replaced with the underscore.	Status	NetworkEntityRuntime MBean

Monitoring the WS SSU

Table 18–16 describes the attributes that enable you to monitor the WS SSU.

Table 18–16 WS SSU Monitoring Attributes

Attribute	Description	Type	MBean
Status	Specifies a network entity status: ■ 0 - Network entity is unavailable ■ 1 - Network entity is available ■ 2 - Status of the network entity is unknown	Status	NetworkEntityRuntime MBean
Value	Specifies the monitored network entity. The attribute contains the address of the network entity in the URI format where with the colon character (:) is replaced with the underscore.	Status	NetworkEntityRuntime MBean

Monitoring the Orchestration Engine

Table 18–17 describes attributes that enable you to monitor the OE.

Table 18–17 Orchestration Engine Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the total number of sessions that the OE handled in the last measurement period	Counter	OeRuntimeMBean
SessionGauge	Returns the number of SIP sessions that the OE is currently handling	Gauge	NOT IMPLEMENTED
SuccessfulApplication TriggeringCount	Returns the number of successful application triggering that the OE performed in the last measurement period	Counter	OeRuntimeMBean
UnsuccessfulApplication TriggeringCount	Returns the number of unsuccessful application triggering that the OE attempted to perform in the last measurement period	Counter	OeRuntimeMBean
SuccessfulUAServerTriggeringCounter	Returns the number of 2xx and 3xx responses that the OE received in the last measurement period	Counter	OeRuntimeMBean
ExecutionCount	Returns the number of times that the OE triggered the specific OLP in the last measurement period	Counter	OlpRuntimeMBean

Table 18–17 (Cont.) Orchestration Engine Monitoring Attributes

Attribute	Description	Type	MBean
SuccessfulQueryCount	Returns the number of successful queries that an OPR executed in the last measurement period	Counter	OprRuntimeMBean
UnsuccessfulQueryCount	Returns the number of unsuccessful queries that an OPR attempted to execute in the last measurement period	Counter	OprRuntimeMBean

Monitoring IM-SCF

You can monitor the following interfaces of IM-SCF:

- CAP
- INAP CS-1
- WIN
- AIN
- TCAP

Table 18–18 describes attributes that enable you to monitor the CAP, INAP CS-1, WIN, and AIN interfaces of IM-SCF. For more information on monitoring the TCAP interface, see "Monitoring the TCAP Interface".

Table 18–18 IM-SCF Monitoring Attributes

Attribute	Description	Type	MBean
FullControlCount	Returns the number of fully controlled sessions that IM-SCF handled in the last measurement period. Full control means that IM-SCF handled the session during the session setup and continues to handle this session after the session is established.	Counter	ImscfCapRuntimeMBean ImscfWinRuntimeMBean ImscfInapCs1Runtime MBean ImscfAinRuntimeMBean
InitialControlCount	Returns the number of non-fully controlled sessions. Non-full control means that IM-SCF handled the session only during the session setup.	Counter	ImscfCapRuntimeMBean ImscfWinRuntimeMBean ImscfInapCs1Runtime MBean ImscfAinRuntimeMBean
SessionCount	Returns the total number of sessions that IM-SCF handled in last measurement period.	Counter	ImscfCapRuntimeMBean ImscfWinRuntimeMBean ImscfInapCs1Runtime MBean ImscfAinRuntimeMBean
SessionGauge	Returns the total number of sessions that IM-SCF is currently handling.	Counter	NOT IMPLEMENTED

Table 18–18 (Cont.) IM-SCF Monitoring Attributes

Attribute	Description	Type	MBean
RcvCount	Returns the number of messages that IM-SCF received from the network in the last measurement period.	Counter	MessageByOpRuntime MBean InitialDpByEventType RuntimeMBean ErbByEventTypeRuntime MBean InitialDpByServiceKey RuntimeMBean InitialDpSmsByService KeyRuntimeMBean
SndCount	Returns the number of messages that IM-SCF sent to the network in the last measurement period.	Counter	MessageByOpRuntime MBean InitialDpByEventType RuntimeMBean ErbByEventTypeRuntime MBean InitialDpByServiceKey RuntimeMBean InitialDpSmsByService KeyRuntimeMBean

Monitoring IM-SSF

You can monitor the following interfaces of IM-SSF:

- CAP
- INAP CS-1
- WIN
- AIN
- TCAP

Table 18–19 describes attributes that enable you to monitor the CAP, INAP CS-1, WIN, and AIN interfaces of IM-SSF. For more information on monitoring the TCAP interface, see "Monitoring the TCAP Interface".

Table 18–19 IM-SSF Monitoring Attributes

Attribute	Description	Type	MBean
FullControlCount	Returns the number of fully controlled sessions that IM-SSF handled in the last measurement period. Full control means that IM-SSF handled the session during the session setup and continues to handle this session after the session is established.	Counter	ImssfCapRuntimeMBean ImssfWinRuntimeMBean ImssfInapCs1Runtime MBean ImssfAinRuntimeMBean

Table 18–19 (Cont.) IM-SSF Monitoring Attributes

Attribute	Description	Type	MBean
InitialControlCount	Returns the number of non-fully controlled sessions. Non-full control means that IM-SSF handled the session during the session setup only.	Counter	ImssfCapRuntimeMBean ImssfWinRuntimeMBean ImssfInapCs1Runtime MBean ImssfAinRuntimeMBean
SessionCount	Returns the total number of sessions that IM-SSF handled in last measurement period.	Counter	ImssfCapRuntimeMBean ImssfWinRuntimeMBean ImssfInapCs1Runtime MBean ImssfAinRuntimeMBean
SessionGauge	Returns the total number of sessions that IM-SSF CAP is currently handling.	Gauge	NOT IMPLEMENTED
UserInteractionCount	Returns the number of sessions that included interaction with a media resource.	Counter	ImssfCapRuntimeMBean
TssfExpiryCount	Returns the number of times that Tssf expired.	Counter	ImssfCapRuntimeMBean
OSideSessionCount	Returns the number of sessions that triggered a service on the O-side.	Counter	ImssfCapRuntimeMBean
TSideSessionCount	Returns the number of sessions that triggered a service on the T-side.	Counter	ImssfCapRuntimeMBean
RcvCount	Returns the number of messages that IM-SSF received from the network in the last measurement period.	Counter	MessageByOpRuntimeMBean InitialDpByEventTypeRuntimeMBean ErbByEventTypeRuntimeMBean InitialDpByServiceKeyRuntimeMBean InitialDpSmsByServiceKeyRuntimeMBean
SndCount	Specifies the number of messages that IM-SSF sent to the network in the last measurement period.	Counter	MessageByOpRuntimeMBean InitialDpByEventTypeRuntimeMBean ErbByEventTypeRuntimeMBean InitialDpByServiceKeyRuntimeMBean InitialDpSmsByServiceKeyRuntimeMBean

Monitoring IM-ASF

Table 18–20 describes attributes that enable you to monitor IM-ASF.

Table 18–20 IM-ASF Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the number of sessions that IM-ASF SIP handled	Counter	ImasfSipRuntimeMBean

Monitoring R-IM-ASF

Table 18–21 describes attributes that enable you to monitor R-IM-ASF.

Table 18–21 R-IM-ASF Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the number of sessions that R-IM-ASF SIP handled.	Counter	RimasfSipRuntimeMBean

Monitoring IM-OCF

You can monitor the following:

- Root ImocfRuntimeMBean
- Diameter interface

Table 18–22 describes attributes that enable you to monitor the root ImocfRuntimeMBean. For more information on monitoring the Diameter interface, see "Monitoring the Diameter Interface".

Table 18–22 IM-OCF Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the number of sessions that IM-OCF handled	Counter	ImocfRuntimeMBean
SessionGauge	Returns the number of sessions that IM-OCF is currently handling	Gauge	NOT IMPLEMENTED
AbandonSessionCount	Returns the number of session establishing attempts that were abandoned by a user during setup in the last measurement period	Counter	ImocfRuntimeMBean
AverageCallDuration	Returns the average call duration in the im-ocf	Counter	NOT IMPLEMENTED
ErrorSessionCount	Returns the number of attempt session establishment that failed during setup in the last measurement period	Counter	NOT IMPLEMENTED

Table 18–22 (Cont.) IM-OCF Monitoring Attributes

Attribute	Description	Type	MBean
ServiceNotApproved SessionCount	Returns the number of attempt session establishment that were not approved by the service (no credit) in the last measurement period	Counter	NOT IMPLEMENTED
UserTerminatedSession Count	Returns the number of established sessions that were terminated by one of the users in the last measurement period	Counter	ImocfRuntimeMBean
ServiceTerminatedSession Count	Returns the number of established sessions that were terminated by the service in the last measurement period	Counter	ImocfRuntimeMBean

Monitoring IM-OCF PCP

You can monitor the following:

- Root ImocfRuntimeMBean
- Diameter interface

Table 18–23 describes attributes that enable you to monitor the root ImocfpcpRuntimeMBean. For more information on monitoring the Diameter interface, see "Monitoring the Diameter Interface".

Table 18–23 IM-OCF PCP Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the number of sessions that IM-OCF PCP handled	Counter	ImocfpcpRuntimeMBean
SessionGauge	Returns the number of sessions that IM-OCF PCP is currently handling	Gauge	NOT IMPLEMENTED
AbandonSessionCount	Returns the number of session establishing attempts that were abandoned by a user during setup in the last measurement period	Counter	ImocfpcpRuntimeMBean
AverageCallDuration	Returns the average call duration in the IM-OCF	Counter	NOT IMPLEMENTED
ErrorSessionCount	Returns the number of attempt session establishment that failed during setup in the last measurement period	Counter	NOT IMPLEMENTED

Table 18–23 (Cont.) IM-OCF PCP Monitoring Attributes

Attribute	Description	Type	MBean
ServiceNotApproved SessionCount	Returns the number of attempt session establishment that were not approved by the service (no credit) in the last measurement period	Counter	NOT IMPLEMENTED
UserTerminatedSession Count	Returns the number of established sessions that were terminated by one of the users in the last measurement period	Counter	ImocfpcpRuntimeMBean
ServiceTerminatedSession Count	Returns the number of established sessions that were terminated by the service in the last measurement period	Counter	ImocfpcpRuntimeMBean

Monitoring R-IM-OCF

You can monitor the following:

- Root ImocfRuntimeMBean
- Diameter interface

Table 18–24 describes attributes that enable you to monitor the root RimocfRuntimeMBean. For more information on monitoring the Diameter interface, see "Monitoring the Diameter Interface".

Table 18–24 R-IM-OCF Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the number of sessions that R-IM-OCF handled in the last measurement period	Counter	RimocfRuntimeMBean
SessionGauge	Returns the number of sessions that R-IM-OCF is currently handling	Gauge	RimocfRuntimeMBean
AbandonSessionCount	Returns the number of session establishing attempts that were abandoned by a user during setup in the last measurement period	Counter	RimocfRuntimeMBean
UserTerminatedSession Count	Returns the number of established sessions that were terminated by one of the users in the last measurement period	Counter	RimocfRuntimeMBean
ServiceTerminatedSession Count	Returns the number of established sessions that were terminated by the service in the last measurement period	Counter	RimocfRuntimeMBean

Monitoring IM-OFCF PCP

You can monitor the following:

- Root ImOfcfRuntimeMBean
- Rf interface
- Diameter interface

Table 18–25 describes attributes that enable you to monitor IM-OFCF PCP.

Table 18–25 IM-OFCF PCP Monitoring Attributes

Attribute	Description	Type	MBean
ImofcfSessionCount	Returns the number of sessions that IM-OFCF PCP handled	Counter	ImofcfRuntimeMBean
RequestCount	Returns the number of ACRs that IM-OFCF PCP sends to the Oracle BRM application through the Diameter interface	Counter	DiameterRuntimeMBean
AnsCount	Returns the number of ACAs that IM-OFCF PCP received from the Oracle BRM application through the Diameter interface	Counter	DiameterRuntimeMBean
SessionCount	Returns the number of sessions that IM-OFCF PCP handled through the Rf interface	Counter	RfRuntimeMBean
AcrCount	Returns the number of ACRs that IM-OFCF PCP sends to the Oracle BRM application through the Rf interface	Counter	RfRuntimeMBean
StartAcrCount	Returns the number of ACRs that IM-OFCF PCP received with the Record-Type AVP set to "START"	Counter	RfRuntimeMBean
InterimAcrCount	Returns the number of ACRs that IM-OFCF PCP received with the Record-Type AVP set to "INTERIM"	Counter	RfRuntimeMBean
StopAcrCount	Returns the number of ACRs that IM-OFCF PCP received with the Record-Type AVP set to "STOP"	Counter	RfRuntimeMBean
EventAcrCount	Returns the number of ACRs that IM-OFCF PCP received with the Record-Type AVP set to "EVENT"	Counter	RfRuntimeMBean

Table 18–25 (Cont.) IM-OFCF PCP Monitoring Attributes

Attribute	Description	Type	MBean
AcaCount	Returns the number of ACAs that IM-OFCF PCP received from the Oracle BRM application through the Rf interface	Counter	RfRuntimeMBean
AcaSuccessCount	Returns the number of ACAs that IM-OFCF PCP received with the Result-Code AVP set to a value less than 3000 (that is success)	Counter	RfRuntimeMBean
AcaErrorCount	Returns the number of ACAs that IM-OFCF PCP received with the Result-Code AVP set to a value equals or greater than 3000 (that is failure)	Counter	RfRuntimeMBean

Monitoring R-IM-OFCF Radius

You can monitor the following:

- Root RimOfcfRuntimeMBean
- Rf interface
- Diameter interface

Table 18–26 describes attributes that enable you to monitor R-IM-OFCF Radius.

Table 18–26 R-IM-OFCF Radius Monitoring Attributes

Attribute	Description	Type	MBean
RimofcfSessionCount	Returns the number of sessions that R-IM-OFCF Radius handled	Counter	RimOfcfRuntimeMBean
RequestCount	Returns the number of ACRs that R-IM-OFCF Radius sends to the Oracle BRM application through the Diameter interface	Counter	DiameterRuntimeMBean
AnsCount	Returns the number of ACAs that R-IM-OFCF Radius received from the Oracle BRM application through the Diameter interface	Counter	DiameterRuntimeMBean
SessionCount	Returns the number of sessions that R-IM-OFCF Radius handled through the Rf interface	Counter	RfRuntimeMBean
AcrCount	Returns the number of ACRs that R-IM-OFCF Radius sends to the Oracle BRM application through the Rf interface	Counter	RfRuntimeMBean

Table 18–26 (Cont.) R-IM-OFCF Radius Monitoring Attributes

Attribute	Description	Type	MBean
StartAcrCount	Returns the number of ACRs that R-IM-OFCF Radius received with the Record-Type AVP set to "START"	Counter	RfRuntimeMBean
InterimAcrCount	Returns the number of ACRs that R-IM-OFCF Radius received with the Record-Type AVP set to "INTERIM"	Counter	RfRuntimeMBean
StopAcrCount	Returns the number of ACRs that R-IM-OFCF Radius received with the Record-Type AVP set to "STOP"	Counter	RfRuntimeMBean
EventAcrCount	Returns the number of ACRs that R-IM-OFCF Radius received with the Record-Type AVP set to "EVENT"	Counter	RfRuntimeMBean
AcaCount	Returns the number of ACAs that R-IM-OFCF Radius received from the Oracle BRM application through the Rf interface	Counter	RfRuntimeMBean
AcaSuccessCount	Returns the number of ACAs that R-IM-OFCF Radius received with the Result-Code AVP set to a value less than 3000 (that is success)	Counter	RfRuntimeMBean
AcaErrorCount	Returns the number of ACAs that R-IM-OFCF Radius received with the Result-Code AVP set to a value equals or greater than 3000 (that is failure)	Counter	RfRuntimeMBean

Monitoring IM-PSX

You can monitor the following interfaces of IM-PSX:

- MAP (for GSM networks) or ANSI-MAP (for CDMA networks)
- TCAP

Table 18–27 describes attributes that enable you to monitor the MAP interface. For more information on monitoring the TCAP interface, see "Monitoring the TCAP Interface".

Table 18–27 IM-PSX Monitoring Attributes

Attribute	Description	Type	MBean
NetworkInitiatedDialog Count	Returns the number of non-module-initiated dialogs that the module handled in the last measurement period	Counter	DialogRuntimeMBean
ServiceInitiatedDialog Count	Returns the number of dialogs that a module initiated and handled in the last measurement period	Counter	DialogRuntimeMBean
RcvCount	Returns the number of messages that a module received from the network in the last measurement period	Counter	MessageRuntimeMBean
SndCount	Returns the number of messages that a module sent to the network in the last measurement period	Counter	MessageRuntimeMBean MessageByOpRuntimeMBean

Monitoring IM-PSX Plugin

Table 18–28 describes attributes that enable you to monitor IM-PSX Plugin.

Table 18–28 IM-PSX Plugin Monitoring Attributes

Attribute	Description	Type	MBean
SessionCount	Returns the total number of sessions that the module handled in the last measurement period	Counter	ImtcapRuntimeMBean
ASInitiatedCount	Returns the number of sessions initiated by a SIP application that the module handled in the last measurement period	Counter	ImtcapRuntimeMBean
NetworkInitiatedCount	Returns the number of sessions initiated by the network that the module handled in the last measurement period	Counter	ImtcapRuntimeMBean

Monitoring IM-UIX-USSD

Table 18–29 describes attributes that enable you to monitor IM-UIX-USSD.

Table 18–29 IM-UIX-USSD Monitoring Attributes

Attribute	Description	Type	MBean
MSInitiatedSessionCount	Returns the number of USSD sessions initiated by a mobile subscriber.	Counter	ImuixUssdGsmMap3 RuntimeMBean
ServiceInitiatedSession Count	Returns the number of USSD sessions initiated by a SIP application.	Counter	ImuixUssdGsmMap3 RuntimeMBean

Table 18–29 (Cont.) IM-UIX-USSD Monitoring Attributes

Attribute	Description	Type	MBean
MSInitiatedSessionBy ServiceCodeRcvCount	Returns the number of processUnstructured-Request messages that IM-UIX-USSD received from an HLR and classified by the service code.	Counter	ImuixUssdGsmMap3 RuntimeMBean

Monitoring IM-UIX-SMS

Table 18–30 describes the attributes that enable you to monitor IM-UIX-SMS.

Table 18–30 IM-UIX-SMS Monitoring Attributes

Attribute	Description	Type	MBean
NetworkInitiatedSession Count	Returns the number of SMS sessions initiated by the network.	Counter	ImUixSmsSmpp34 RuntimeMBean
ApplicationInitiated SessionCount	Returns the number of SMS sessions initiated by the application.	Counter	ImUixSmsSmpp34 RuntimeMBean
SmscDeliveryReceiptRcv Count	Returns the number SMSC delivery receipts which are carried over deliver_sm operations, that IM-UIX-SMS received	Counter	ImUixSmsSmpp34 RuntimeMBean
SmsOverDeliverSmRcv Count	Returns the number of delivery_sm operations with the Message Type parameter set to 0.	Counter	ImUixSmsSmpp34 RuntimeMBean
SmeDeliveryAckRcvCount	Returns the number of Delivery ACK messages which are carried over deliver_sm operations, that IM-UIX-SMS received from an SME	Counter	ImUixSmsSmpp34 RuntimeMBean
SmeUserManualAckRcv Count	Returns the number of User/Manual ACK messages which are carried over deliver_sm operations, that IM-UIX-SMS received from an SME	Counter	ImUixSmsSmpp34 RuntimeMBean
SessionCount	Returns the total number of sessions that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
RequestCount	Returns the total number of requests that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean

Table 18–30 (Cont.) IM-UIX-SMS Monitoring Attributes

Attribute	Description	Type	MBean
ResponseCount	Returns the total number of responses that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
SubmitSmCount	Returns the total number of submit_sm operations that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
DeliverSmCount	Returns the total number of deliver_sm operations that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
SubmitSmRespCount	Returns the total number of submit_sm_resp operations that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
DeliverSmRespCount	Returns the total number of deliver_sm_resp operations that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
SubmitSmRespSuccess Count	Returns the number of submit_sm success responses that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
SubmitSmRespError Count	Returns the number of submit_sm error responses that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
DeliverSmRespSuccess Count	Returns the number of deliver_sm success responses that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean
DeliverSmRespError Count	Returns the number of deliver_sm error responses that IM-UIX-SMS handled in the last measurement period.	Counter	SmppRuntimeMBean

Monitoring SM-PME

Table 18–31 describes attributes that enable you to monitor the IN interface

Table 18–31 SM-PME Monitoring Attributes

Attribute	Description	Type	MBean
successfulMappingCount	Specifies the number of successful mapping executed	Counter	PmeRuntimeMBean
failMappingCount	Specifies the number of failed mapping	Counter	PmeRuntimeMBean

19

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.

Accessing the Service Broker MIB

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.

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.

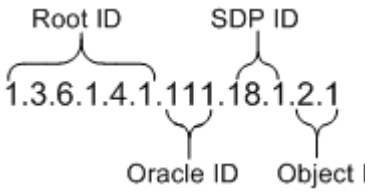
The MIB module files are included in the Oracle Communications Service Broker Media Pack.

> **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 19–1 shows the elements of ab OID.

Figure 19–1 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 Service Broker MIB modules 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. 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 user interface to configure the SNMP operating behavior of the deployment.

The settings include the general SNMP settings for the Managed Servers and trap destinations and settings, as described in the following sections.

Accessing SNMP Configuration Settings

To access the SNMP configuration settings in the Administration Console interface, 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 19–1 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 19–1.

Table 19–1 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.
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 19–2 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.
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.
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 toSNMPv3.
loggingLevel	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.

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 19–3 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.
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 19–4 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).
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.

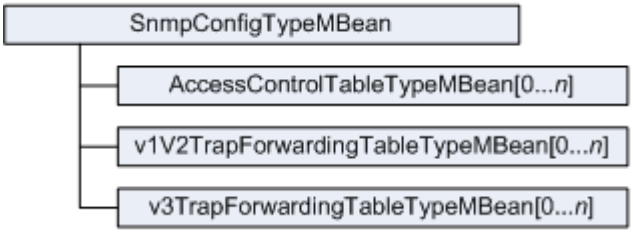
The new trap forwarding settings appear in the table.

Configuring SNMP with Java MBeans

Service Broker provides a set of MBeans that expose attributes and operations for configuring SNMP through JMX. This section describes the MBeans and settings you use to configure the SNMP service.

Figure 19–2 shows the hierarchy of the SNMP configuration MBeans. Under the root configuration MBean are MBeans for configuring access control tables, SNMPv1 and SNMPv2c trap destinations, and SNMPv3 trap destinations.

Figure 19–2 SNMP Configuration MBean Hierarchy



To configure the Service Broker SNMP agent, follow these general steps:

1. Access **SnmpConfigTypeMBean** using a JMX client., to define a managed agent.
2. For each Processing or Signaling Server, invoke the **addTarget** operation.
3. In each new target instance, define the agent properties, including listening port, SNMP version, and logging level. At a minimum, enable the agent by setting its **enabled** attribute to **true**.
4. If desired, specify access control restrictions applicable to SNMP queries using **AccessControlTableTypeMBean**. The MBean includes the community string required for access and IP filtering settings.
5. Specify trap destinations using **v1V2TrapForwardingTableTypeMBean** or **v3TrapForwardingTableTypeMBean**. If desired, add multiple destinations, each with its own security properties or SNMP version.

The following sections provide reference information on the MBeans.

SnmpConfigTypeMBean

SnmpConfigTypeMBean is the root MBean for SNMP configuration. This MBean holds the basic configuration settings for the agent.

JAR File

oracle.axia.snmp-*version*.jar

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

Package

oracle.axia.config.beans.snmp

Object Name

oracle:name=oracle.axia.snmp,name0=SnmpConfig,type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0

Factory Method

Created automatically.

Number of Allowed Occurrences

Maximum: 1

Attributes

- enabled
- port
- version
- loggingLevel

For more information about these parameters, see Table 19–1.

Operations

You can add or remove instances of **AccessControlTableMBean**, **v1V2TrapForwardingTableMBean**, and **v3TrapForwardingTableMBean** using the following operations.

addAccessControlTableType

Creates an instance of **AccessControlTableTypeMBean**. The access control table contains authentication settings for SNMP query requests.

removeAccessControlTable

Removes an instance of **AccessControlTableMBean**.

addV1V2TrapForwardingTable

Creates an instance of **v1V2TrapForwardingTableMBean**. The MBean contains settings that control the SNMPv1 and SNMPv2c traps sent by the Service Broker agent.

removeV1V2TrapForwardingTable

Removes an instance of **v1V2TrapForwardingTableMBean**.

addV3TrapForwardingTable
Creates an instance of **v3TrapForwardingTableMBean**. The MBean contains settings that control the SNMPv1 and SNMPv2c traps sent by the Service Broker agent.

removeV3TrapForwardingTable
Removes an instance of **v3TrapForwardingTableMBean**.

AccessControlTableTypeMBean

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

JAR File

oracle.axia.snmp-*version*.jar

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

Package

oracle.axia.config.beans.snmp

Object Name

oracle:name=oracle.axia.snmp,name0=SnmpConfig,name1=snmpSet[*n*],type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0

where *n* is an index.

Factory Method

SnmpConfigType.addAccessControlTableType()

Number of Allowed Occurrences

Minimum: 0; No maximum

Attributes

- aclAccess
- aclCommunity
- aclManagers

For more information about these parameters, see Table 19–2.

Operations

None

v1V2TrapForwardingTableTypeMBean

v1V2TrapForwardingTableTypeMBean specifies trap forwarding settings for SNMPv1 or SNMPv2c trap destinations. You must configure this MBean for each SNMP manager to which you want to send SNMPv1 or SNMPv2c traps from the SNMP agent.

JAR File

oracle.axia.snmp-*version*.jar

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

Package

oracle.axia.config.beans.snmp

Object Name

oracle:name=oracle.axia.snmp,name0=SnmpConfig,name1=snmpSet[*n*],type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0

where *n* is an index.

Factory Method

SnmpConfigType.addV1V2TrapForwardingTableType()

Number of Allowed Occurrences

Minimum: 0; No maximum

Attributes

- community
- managerHost
- managerPort
- retries
- timeout
- version

For more information about these parameters, see Table 19–3.

Operations

None

v3TrapForwardingTableTypeMBean

v3TrapForwardingTableTypeMBean specifies destinations for SNMPv3 traps.

SNMPv3 provides enhanced security capabilities over versions 1 or 2 of SNMP. Accordingly, this MBean contains additional security-related settings.

JAR File

oracle.axia.snmp-*version*.jar

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

Package

oracle.axia.config.beans.snmp

Object Name

oracle:name=oracle.axia.snmp,name0=SnmpConfig,name1=snmpSet[*n*],type=oracle.axia.cm.ConfigurationMBean,version=1.0.0.0

where *n* is an index.

Factory Method

SnmpConfigType.addV3TrapForwardingTableType()

Number of Allowed Occurrences

Minimum: 0; No maximum

Attributes

- community
- managerHost
- managerPort
- retries
- securityLevel
- timeout
- userContextName
- userName
- userSecModel
- version

For more information about these parameters, see Table 19–4.

Operations

None

20

Viewing Service Broker SDRs

This chapter explains how you can monitor Oracle Communications Service Broker using Service Broker Service Description Records (SDRs).

Understanding Service Broker Service Description Records

A Service Description Record (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 conventions 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 pre-configured file size, at which time a new file is created. The default file size is 100 KB.

When the number of files reaches a pre-configured 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 pre-configured 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 Description Record Format

Each SDR consists of parameter-value pairs that store information about service activation and delivery performed through Service Broker. Table 20–1 describes the parameters available in each SDR.

Table 20–1 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.
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.

Table 20–1 (Cont.) Service Broker SDR Fields

Field	Description
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 20–2 describes the tags generated by the IM-OCF.

Table 20–2 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.

21

Preventing Service Broker System Overload

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

Understanding Overload Protection

In some cases, such as traffic peaks or unexpected failure of network entities, the load on Service Broker modules can increase significantly. This can cause a situation known as system overload, where Service Broker modules have insufficient resources to handle new sessions. If overload is not handled correctly, the system can crash, and critical data can be lost.

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 and configure how Service Broker behaves if system overload occurs.

To activate overload protection you have to perform the following steps:

1. Identify key overload indicators

   You can choose any of the many counters and gauges provided by Service Broker to serve as key overload indicators. When you later set these gauges and counters to be key overload indicators, Service Broker triggers overload prevention based on the values measured by these gauges and counters.

   It is recommended to identify at least the following two system level measurements as your key overload indicators:

   - **`SystemCountersRuntimeMBean.SessionGauge`**: A gauge measuring the number of active sessions currently handled by Service Broker.
   - **`SystemCountersRuntimeMBean.InitialRequestCount`**: A counter measuring the rate in which Service Broker receives new sessions.

   Note that identifying a counter or a gauge as a key overload indicator does not yet make it a key overload indicator. You define the gauges and counters as key overload indicators in step 3.

2. Define thresholds for the identified indicators

   For each key overload indicator (that is gauge or counter) that you identified in the step 1, you need to define an upper threshold. When a key overload indicator crosses this threshold, Service Broker activates overload protection.

3. Define key overload indicators

You need to define the key overload indicators that you identified in step 1, as valid and active key overload indicators

4. Configure overload protection behavior

   When Service Broker identifies overload (that is, when any of the key overload indicators has crossed its threshold), it continues handling active sessions but stops accepting new sessions. In this step, you can configure how Service Broker responds to SIP and Diameter network entities that attempt to establish sessions during a system overload. For 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.

Configuring Overload Protection

You can configure overload protection using either the Administration Console or the Service Broker configuration MBeans.

For a description of the overload protection configuration, see the discussion on managing the Service Broker processing tier in *Oracle Communications Service Broker Processing Domain Configuration Guide.*

A

System Administrator's Reference

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

Details for Administration Clients

This section specifies the directory structure, directory contents and start-scripts for administration clients.

Directory Structure and Contents for an Administration Client

All administration clients are installed under the directory:

Linux and Solaris: *Oracle_home*/**ocsb60/admin_console**

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/ocsb60

Directory	Description
admin_console	Top-level directory for all administration clients. Contains start-scripts for: ■ Standalone Administration Console ■ Web Administration Console 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_console/applications	Created during start-up. Empty directory.
admin_console/extensions	Extensions to the Administration console specific to the features installed.
admin_console/extensions_cef	Extensions to the Administration console specific to the features installed.

Table A–1 (Cont.) Directory Contents and Structure for Administration Clients Relative to Oracle_home/ocsb60

Directory	Description
admin_console/domain_ configuration	Contains these directories: ■ **/meta** - Contains metadata .xml files that support domain creation. ■ **/domain_configuration** - Supporting files for domain creation.
admin_console/modules	Contains all OSGi bundles for the administration clients, the Processing Server and the Signaling Server.
admin_console/osgi	Contains OSGi-specific configuration for the Administration Console processes.
admin_console/properties	It contains property files used by the start-scripts for: ■ Standalone Administration Console ■ The SVC and VPN applications. ■ Web Administration Console server ■ Scripting Engine ■ Domain Web server ■ Hosted domains
admin_console/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_console/utils	Contains utilities used by the SVC and VPN features.
admin_console/workspace	Contains metadata for administration clients.

Start Scripts

Table A–2 give 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 Scripts for Configuration and Management" for details.
start.sh	Starts the Standalone Administration Console. **start.sh** calls **common.sh**.
web.sh	Starts the Web Administration Console server. **web.sh** calls **common.sh**.

Table A–2 (Cont.) Start-scripts for the Administration Clients

Script	Description
host.sh	Starts the Domain Web server. **host.sh** calls **common.sh**. See "Starting and Stopping Processing Servers and Signaling Servers" for information on how to use the script.
common.sh	Starts the Standalone Administration Console, Web Administration Console 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_console/properties** and their settings.

Table A–3 Property files used by the Administration Clients

Property File	Description
common.properties	Defines properties common to the: ■ Standalone Administration Console ■ Web Administration Console server ■ Scripting Engine ■ Domain Web server The properties specified are: ■ **axia.production.mode** ■ **axia.console.log4j.server.port** ■ **axia.ssl.cipher_suite**s ■ **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–9 for details on the property settings. See Table A–10 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–9 for details on the property settings. See Table A–10 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–9 for details on the property settings. See Table A–10 for information on the security entries.
standalone.properties	Defines properties for the Standalone Administration Console. The properties specified are: ▪ **axia.platform** ▪ **org.eclipse.equinox.http.jetty.http.enabled** ▪ **org.eclipse.equinox.http.jetty.https.enabled** See the **standalone.properties** file and Table A–9 for details on the property settings. See Table A–10 for information on the security entries.
web.properties	Defines properties for the Web Administration Console server. The properties specified are: ▪ **axia.platform** ▪ **axia.require.domain** ▪ **axia.basic.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 **web.properties** file and Table A–9 for details on the property settings. See Table A–10 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/**ocsb60/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/ocsb60

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.

Start Scripts for Processing Servers and Signaling Servers

Table A–5 gives information about start-scripts for Processing Servers and Signaling Servers.

Table A–5 Start-scripts for Processing Servers and Signaling Servers

Script	Description
start.sh	Starts the Standalone Administration Console. Defines the **AXIA_OPTS** and AXIA_MEM_OPTS environment variables.

Properties File for Managed Servers

Table A–6 gives information property files in:

Oracle_home/**managed_server/properties**

Table A–6 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–9 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–7 describes the directory structure and contents of the directory structure.

Table A–7 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–8 gives information about the environment variables used.

Table A–8 Environment variables

Variable	Description
AXIA_OPTS	Defines any additional Java options to use.
AXIA_MEM_OPTS	Overrides the default memory settings for the JVM, such as heap size.

System Properties

Table A–9 describes the general system properties defined for Oracle Communications Service Broker. The security-related property entries are listed in Table A–10.

Table A–9 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**. ▪ **standalone** in **standalone.properties**. ▪ **web** in **web.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.

Table A–9 (Cont.) Description of System Properties

System Property	Description
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.
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: ■ **web.properties** ■ **hosting.properties** The setting in **web.properties** defines the port for the Web Administration Console 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** ■ **standalone.properties** ■ **web.properties** Must always be set to **false** in **script.properties**, **standalone.properties**, and **web.properties**.

Table A–9 (Cont.) Description of System Properties

System Property	Description
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** ■ **standalone.properties** ■ **web.properties** Must always be set to **false** in **script.properties**, **standalone.properties**, and **web.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 **web.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 **web.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.
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–10 lists the security-related property file entries. See Table A–9 for the other system property file entries.

Table A–10 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**.

Table A–10 (Cont.) System Security Properties

System Security Property	Description
axia.basic.auth	Boolean. Specifies if HTTP basic authentication is used when the Web Administration Console connects to the Web Administration Console server. Set this property to: ■ **true** to use HTTP basic authentication. ■ **false** to not use HTTP basic authentication. Set in **web.properties**.
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.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 requried to use SSL security in 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.
axia.ssl.cipher_suites	Specifies the combinations of ciphers that Service Broker supports for SSL communication between the Administration Console 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**

Table A–10 (Cont.) System Security Properties

System Security Property	Description
https.cipherSuites	Specifies the combinations of ciphers that Service Broker supports for HTTPS communication between the Administration Console 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 Web Administration Console and the Web Administration Console 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 **web.properties**. and **hosting.properties**.

Directory Contents and Structure for a Domain Configuration

Table A–11 gives information about the directory structure and contents of a domain configuration.

Table A–11 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.

Table A–11 (Cont.) Directory Structure for a Domain Configuration

Directory	Description
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*/**ocsb60**

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 JDKs Relative to Oracle_home/ocsb60

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/**ocsb60**

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

Table A–13 describes the directory structure and the contents of the directory structure.

Table A–13 Directory Structure for Oracle Universal Installer Relative to Oracle_home/ocsb60

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–14 lists these services.

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

Table A–14 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