Oracle® Communications Service Broker

System Administrator's Guide

Release 5.0

E15183-01

December 2010

Oracle Communications Service Broker System Administrator's Guide, Release 5.0

E15183-01

Contents

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

Related Documents

For more information, see the following documents in the Oracle Communications Service Broker Release 5.0 documentation set:

- *Oracle Communications Service Broker Release 5.0 Release Notes*
- *Oracle Communications Service Broker Release 5.0 Concepts Guide*
- *Oracle Communications Service Broker Release 5.0 Configuration Guide*
- *Oracle Communications Service Broker Release 5.0 System Administrator's Guide*
- *Oracle Communications Service Broker Release 5.0 Integration Guide*

1

Configuring Security

This chapter describes the security model for Oracle Communications Service Broker and how to configure security:

Understanding the Security Model

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

Security Between 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 any administration client to authenticate. Administration clients includes:

- Stand-alone Administration Console
- Web Administration Console server

- Scripting Engine

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

It is recommended to secure the JMX port with SSL encryption and to have client authentication enabled. Set the following domain properties to **true**:

- **axia.ssl**
- **axia.jmx.ssl**
- **axia.jmx.client_auth**

Service Broker uses 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).

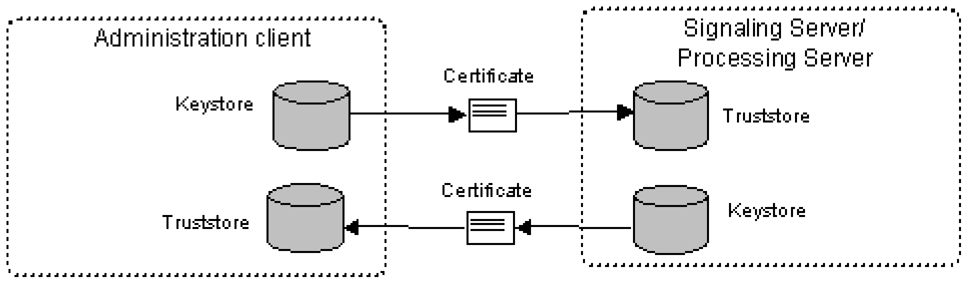
Each server has two stores:

- Server keystore.
- Administration client truststore.

Each administration client has two stores:

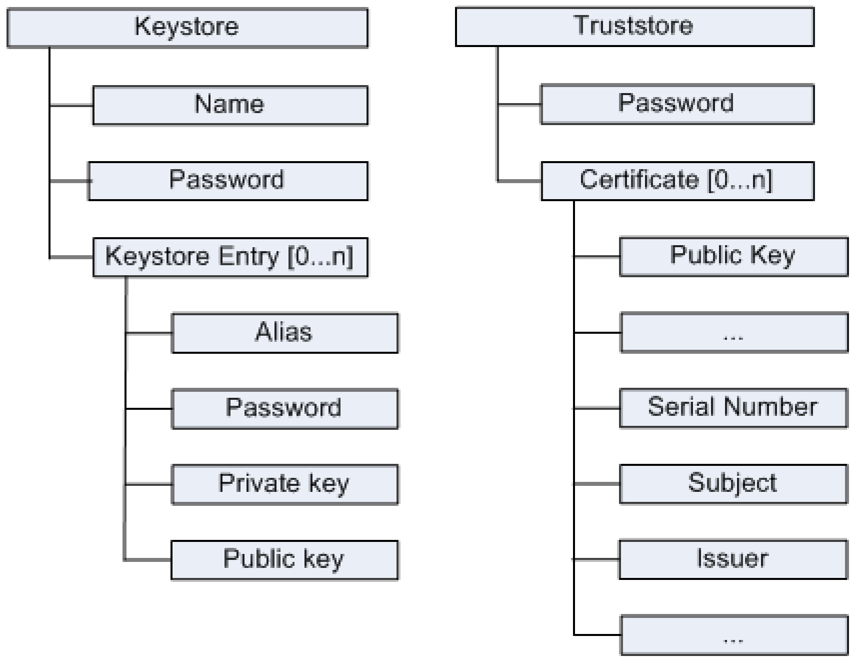
- Administration client keystore.
- Server truststore.

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



A keystore contains keystore entries. A truststore contains public certificates.

Figure 1–2 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.

Security for the Domain Configuration

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:

- Stand-alone 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.

It is recommended to allow only the owner or a very restricted user group to have any privileges to the directory.

When using a hosted domain configuration that is accessed using a Domain Web server you can choose to access it using HTTP or HTTPS. Default is using HTTPS and Basic Authentication. The following properties in *Oracle_home*/**axia/admin_console/properties/hosting.properties** controls the HTTP and HTTPS settings:

- **axia.basic.auth** Set to **true** if Basic Authentication shall be used.
- **org.eclipse.equinox.http.jetty.http.enabled** Set to **true** if HTTP shall be used. Must be set to **false** if HTTPS is used.
- **org.eclipse.equinox.http.jetty.http.port** Set to the port number to use for HTTP connections.
- **org.eclipse.equinox.http.jetty.https.enabled** Set to **true** if HTTPS shall be used. Must be set to **false** if HTTP is used.
- **org.eclipse.equinox.http.jetty.https.port** Set to the port number to use for HTTPS connections.

When HTTPS and Basic Authentication are enabled, keystore passwords and username and passwords for Basic Authentication are prompted for when starting the Domain Web serve and managed servers.

Security Between the Web Administration Console and the Web Administration Console Server

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

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

Security for the Administration Port

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

It is recommended to secure the port with SSL encryption and to have client authentication enabled. Set the following domain properties to **true**:

- **axia.ssl**
- **axia.admin.ssl**
- **axia.admin.client_auth**

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 listen address as specified in it's key.

See "Enabling and Disabling SSL".

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

- If the server name is specified as host name in the domain configuration, one of the following applies to how the 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`

Cluster Security

The Coherence cluster can be secured using standard Coherence configuration settings.

The recommended way to secure the cluster is to create a Coherence configuration override file, package it as a an OSGi bundle fragment and deploy it.

For more information on concepts in the Coherence Security Framework, refer to *Coherence 3.5 User Guide*, section *Security Framework*.

For information on how to create a Coherence configuration override file and details on the available security options, refer to *Coherence 3.5 User Guide*, section *Operational Configuration Elements*.

Both documents are available from *Oracle Coherence Knowledge Base*:

`http://coherence.oracle.com/display/COH35UG/Coherence+3.5+Home`

The Coherence operational override configuration file must be named **tangosol-coherence-override-axia.xml** in order to complement, rather than replace, the override settings already defined by Service Broker.

For information on how to create an OSGi bundle fragment, refer to *OSGi Service Platform Release 4 Core specification*, section *3.14 Fragment Bundles*. The specification can be downloaded from:

`http://www.osgi.org/Release4/Download`

The fragment host for the OSGi fragment bundle must be **oracle.axia.storage.provider.coherence**.

Password Strength Validation

By default all passwords used for the Web Administration Console and the Domain Web server must be at least 6 characters long, contain at least one lower case character, one upper case character, and one digit.

The strength of the passwords can be configured. The following properties in *Oracle_home***/axia/admin_console/properties/common.properties** controls the password strength:

- **axia.console.password.validation.enabled** Set to **true** to enable password strength validation.
- **axia.console.password.validation.min_length** Defines the minimum password length.
- **axia.console.password.validation.require_lower** Set to **true** to enforce that at least one character in the password must be in lower case.
- **axia.console.password.validation.require_upper** Set to **true** to enforce that at least one character in the password must be in upper case.
- **axia.console.password.validation.require_digit** Set to **true** to enforce that at least one character in the password must be a digit.

About Keytool and X.500 Distinguished Names

The Public Key infrastructure (PKI) is setup using keytool, a key and certificate management tool from SUN. It is a part of the JDK and is found in the sub-directory **bin** of your Java installation. See:

`http://download.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*

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 Public Key Infrastructure Between Administration Clients, Processing Servers, and Signaling Servers

Following is a description on how to use keytool to set up the public key infrastructure (PKI) between administration clients, Processing Servers, and Signaling Servers using self-signed X.500 certificates.

To setup 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. It is suggested 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. It is suggested 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. It is suggested 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 shall be imported to each server's truststore. It is suggested 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 may be in different directories and may have a different names than specified below. The directories and filenames below are according to default settings. See "Defining Keystores and Truststores".

   a. Identify which keystore-truststore pair that 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/**axia/managed_server/serverkeystore**

      Copy or use FTP to put the truststore in:

      Oracle_home/**axia/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/**axia/admin_console/clientkeystore**

      Copy or use FTP to put the truststore in:

      Oracle_Home/**axia/admin_console/clienttruststore**

Enabling and Disabling SSL

Note: It is not recommended to 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.admin.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. This property is valid only if **axia.ssl** is set to **true**.

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.

The properties specified in the domain configuration. See Chapter 6, "Managing Domains".

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

Oracle_home/**axia/managed_server**

Oracle_home/**axia/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 section "System Properties" in Appendix A, "System Administrator’s Reference".

Defining the Keystore for SSL Connections Between a Web Administration Console and a Web Administration Console Server

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 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. It is suggested 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 directory to:

   Oracle_home/**axia/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.

2 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
- Stopping a Processing Server or a Signaling Server
- Starting and Stopping the SS7 Process

Starting a Processing Server or a Signaling Server

To start a Processing Server or a Signaling Server:

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

   Oracle_home/**axia/managed_server**

   Where *Oracle_home* is the Oracle Home directory you defined when you installed the product.
3. Enter:

   ./start.sh *Server Domain_URI*

   where

   - *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://**.

   Example where the domain configuration is accessed using the Domain Web server:

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

   Example where 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 as when you started the domain.

Stopping a Processing Server or a Signaling Server

To stop a Processing Server or a Signaling Server, use the Administration Console. For information on how to stop a server. See *Oracle Communications Service Broker Configuration Guide*.

You can also use the Scripting Engine and the MBean **ManagementAgentMBean**. See example below.

Example 2–1 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>
```

Starting and Stopping the SS7 Process

To connect to the SS7 network, an SS7 process needs to be started on the Signaling Servers where the SS7 SSUs are executing.

The binaries for starting the SS7 process are located in the directory:

Oracle_home/**axia/managed_server/ss7stack**

Where *Oracle_home* is the Oracle home directory.

To start the SS7 process used for connecting to the SS7 network using TDM, enter:

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*

Where:

- *port_number* is the port the SS7 SSU listens to messages from the SS7 process.
- *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.

To start the SS7 process used for connecting to the SS7 network using SIGTRAN, enter:

Linux: **convergin.ss7stack.sigtran.linux** *port_number*

Solaris: **convergin.ss7stack.sigtran.solaris** *port_number*

Where *port_number* is the port the SS7 SSU listens to messages from the SS7 process.

Stop the SS7 process using operating-system commands to terminate it, for example **kill**.

3 Starting and Stopping the Stand-alone Administration Console

This chapter describes how to start and stop the Stand-alone Administration Console:

- Starting the Stand-Alone Administration Console
- Stopping the Stand-Alone Administration Console

Starting the Stand-Alone Administration Console

To start the Stand-alone Administration Console:

1. Login to the physical machine where the Stand-alone 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 directory to:

    *Oracle_home***/axia/admin_console**

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

3. Enter:

    ./start.sh *Domain_configuration_directory*

    Where *Domain_configuration_directory* is the path to the domain configuration directory.

Stopping the Stand-Alone Administration Console

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

4 Starting and Stopping Web Administration Console Servers

This chapter describes how to start and stop Web Administration Console servers:

- Starting the Web Administration Console Server
- Stopping the Web Administration Console Server
- Logging In to the Web Administration Console
- About Setting Up Security for the Web Administration Console Server Security

Starting the Web Administration Console Server

To start a Web Administration Console server:

1. Login 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 directory to:

    Oracle_home/**axia/admin_console**

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

3. Enter:

    ./web.sh *Domain_configuration_directory*

    where *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**

   where

   - **https** or **http** depends on your security configuration. See "Enabling and Disabling SSL" in Chapter 1, "Configuring Security".
   - *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 username and password.

   The authentication information must exactly match the information provided when the Web Administration Console server was started (see "Starting the Web Administration Console Server").

Stopping the Web Administration Console Server

Stop the Web Administration Console server using the Web Administration Console. See *Oracle Communications Service Broker Configuration Guide*.

About Setting Up Security for the Web Administration Console Server Security

The Web Administration Console authenticates with the Processing Servers and the Signaling Servers. It also authenticates with the Web Administration Console.

For information about setting up security for the Web Administration Console Server, see Chapter 1, "Configuring Security."

5 Starting and Stopping Domain Web Servers

This chapter describes how to start and stop Domain Web servers:

- Starting the Domain Web Server
- Stopping the Domain Web Server
- Setting Up Security for the Domain Web Server

Starting the Domain Web Server

The Domain Web server provides Processing Servers and Signaling Servers with HTTP access to the domain configuration and the OSGi bundles for the domain.

It is a Web server that gives HTTP access to the domain configuration, and performs no other functions. The domain configuration must have been created so it can be accessed using the Domain Web server, see Chapter 6, "Managing Domains".

Note: The Domain Web server is intended for test and evaluation purposes only. In production deployments, the domain configuration should be accessed using a shared file system.

To start the Domain Web server:

1. Open a command line shell.

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

2. Change directory to:

   Oracle_home/**axia/admin_console**

   Where *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, Processing Servers and Signaling Servers can access the domain configuration over HTTP or HTTPS.

The port is defined in the property **org.eclipse.equinox.http.jetty.http.port**. Default value is 9000. See Appendix A, "System Administrator's Reference".

Stopping the Domain Web Server

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

Setting Up Security for the Domain Web Server

The Domain Web server must be started by a user that has read privileges to the Domain Configuration directory. See Chapter 1, "Configuring Security".

6 Managing Domains

This chapter describes how to manage and configure a domain.

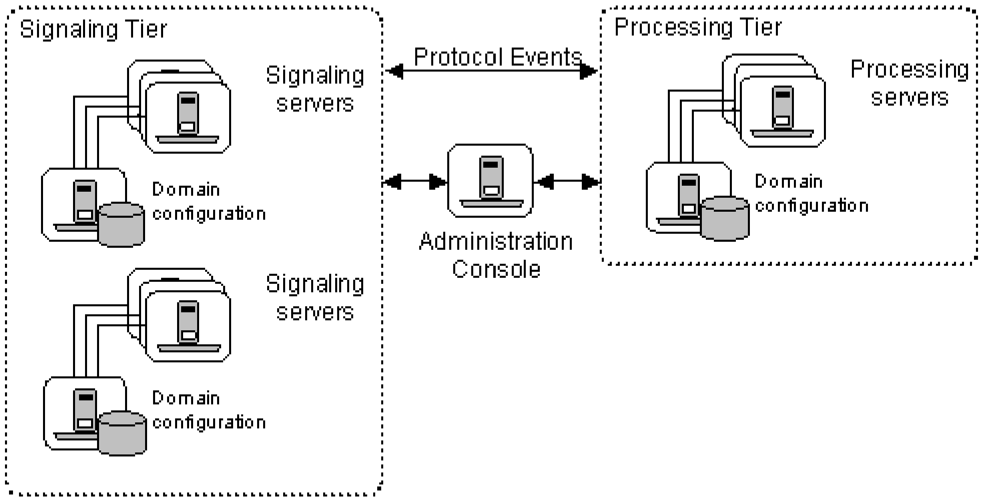
Understanding Domains

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, which means that they all have the same bundles deployed and started. A domain has an associated domain configuration. The domain configuration is kept in a directory that is accessed by all servers in the domain. The directory is accessed using a shared file system or by way of the Domain Web server.

The domains interwork and propagate protocol events across the tier boundaries. Figure 6–1 gives an overview of the tiers and the domains and how they interwork.

Figure 6–1 Overview of Domains and Tiers



An administration client, for example the Administration Console, has access to all domain configurations and can propagate management operations and configuration updates to all Processing Servers and Signaling Servers, regardless of which domain they belong to. An administration client is either the stand-alone Administration Console, the Web Console Web server, the Scripting Engine, or an external client.

The administration clients use the **DomainServiceMBean** to:

- Create and remove a domain configuration.
- Add servers to and remove servers from the domain configuration.
- Manage properties for a domain.
- Manage settings for servers in a domain.
- Specify whether to work in online mode (where configuration updates are propagated to all servers in the domain as each change is made) or offline mode (where updates are saved to the domain configuration directory and propagated to servers later).

Creating a Domain

When you create a domain, the domain's bundles are copied to a domain directory. Configuration data for the particular domain is stored in this directory. You specify the location of the domain directory when you start the servers.

You choose how your servers access the domain configuration directory:

- Using a shared file system

  You must make sure that the servers have access to the specified directory. For information on setting up a shared file system, see the documentation for your operating system.

- Using HTTP

  You must make sure that the Domain Web server can access the domain directory and that the URL is mapped to that directory. See Chapter 5, "Starting and Stopping Domain Web Servers".

Note:

- The Domain Web server is intended for test and evaluation environments only. You should not use the Domain Web server in a production deployment.

In both cases, you need to specify the type of domain you are creating:

- Processing Domain
- Signaling Domain
- Combined

If you need to, you can later change some of the settings (see "Managing Domain Properties").

Two scripts are provided that simplify the domain creation process:

- **create_domain.xml**
- **create_hosted_domain.xml**

The scripts are stored in the following directory:

Oracle_home/**axia/admin_console/scripts**

Execute the scripts using the Scripting Engine. Start the scripts from the directory **admin_console**.

The default cluster setting for domains created by the scripts is IP-multicast, and you will be prompted for multicast settings when running the scripts. This setting can be changed after the domain is created. See Chapter 8, "Managing Clusters" for more information.

For information about the scripts, see "Creating a Domain That Is Accessed Using a Shared File System" and "Creating a Domain That Is Accessed Using HTTP".

You can also use the DomainServiceMBean directly to create your domain:

- To access the domain directory using a Shared File System, invoke the operation void createDomain(String type, String domainPath).
- To access the domain directory using HTTP, invoke the operation void createHostedDomain(String type, String domainPath, String hostAddress).

Creating a Domain That Is Accessed Using a Shared File System

The script **create_domain.xml** creates a domain that is accessed using a shared file system.

To create a domain:

1. Execute the script by entering:

   script.sh scripts/create_domain.xml

2. When you are prompted for **domain.type**, enter one of the following:

   - **ocsb-pn** to create a Processing Domain.
   - **ocsb-ssu** to create a Signaling Domain.
   - **ocsb-basic** to create a combined Domain.

3. When you are prompted for **domain.path**, enter the path to the directory where the domain configuration shall be created.
4. When you are prompted for **domain.axia.ssl**, enter:
   - **true** to enable SSL.
   - **false** to disable SSL.

     This parameter corresponds to the domain property **axia.ssl**. See Managing Domain Properties.
5. When you are prompted for **multicast.address**, enter the IP multicast address to use to join the domains. The value of the **multicast.address** parameter must be within the range 224.0.0.0 to 239.255.255.255.
6. When you are prompted for **multicast.port**, enter the IP multicast port to use to join the domains. The value of the **multicast.port** parameter must be 1024 or greater.
7. When you are prompted for **multicast.ttl**, enter the time-to-live for a multicast. The value of the **multicast.ttl** parameter must be 1 or greater.

Example with input in bold:

```
Enter a value for parameter 'domain.type': ocsb-basic
Enter a value for parameter 'domain.path': /domains/ocsb-basic-fs
Enter a value for parameter 'domain.axia.ssl': false
Enter a value for parameter 'multicast.address': 239.255.255.255
Enter a value for parameter 'multicast.port': 1234
Enter a value for parameter 'multicast.ttl': 1
```

Creating a Domain That Is Accessed Using HTTP

The script **create_hosted_domain.xml** creates a domain that is accessed using HTTP.

To create a domain:

1. Execute the script by entering:

   script.sh scripts/create_hosted_domain.xml
2. When you are prompted for **domain.type**, enter:
   - **ocsb-pn** to create a Processing Domain.
   - **ocsb-ssu** to create a Signaling Domain.
   - **ocsb-basic** to create a combined Domain.
3. When you are prompted for **domain.path**, enter the path to the directory where the domain configuration shall be created.
4. When you are prompted for **domain.hosted.url**, enter the host name and port for the Domain Web server that gives access to the domain configuration.

   The port is defined in the property **org.eclipse.equinox.http.jetty.http.port** in the file **hosting.properties**. Default value is 9000. See Appendix A, "System Administrator's Reference".

   Also see Chapter 5, "Starting and Stopping Domain Web Servers".
5. When you are prompted for **domain.axia.ssl**, enter:
   - **true** to enable SSL.
   - **false** to disable SSL.

This parameter corresponds to the domain property **axia.ssl**. See Managing Domain Properties.

6. When you are prompted for **multicast.address**, enter the IP multicast address to use to join the domains. The value of the **multicast.address** parameter must be within the range 224.0.0.0 to 239.255.255.255.
7. When you are prompted for **multicast.port**, enter the IP multicast port to use to join the domains. The value of the **multicast.port** parameter must be 1024 or greater.
8. When you are prompted for **multicast.ttl**, enter the time-to-live for a multicast. The value of the **multicast.ttl** parameter must be 1 or greater.

Example with input in bold:

```
Enter a value for parameter 'domain.type': ocsb-basic
Enter a value for parameter 'domain.path': /domains/ocsb-basic-http
Enter a value for parameter 'domain.hosted.url': http://localhost:9001/
Enter a value for parameter 'domain.axia.ssl': false
Enter a value for parameter 'multicast.address': 239.255.255.255
Enter a value for parameter 'multicast.port': 1234
Enter a value for parameter 'multicast.ttl': 1
```

Opening a Domain Configuration and Locking It for Changes

You need to open a domain configuration in order to make configuration updates to it. When you open the domain, it is locked for edits from other administration clients. To open a domain configuration, invoke the operation **openDomain** on the MBean **DomainServiceMBean**.

Once you open the domain, you can make configuration changes in two 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. To change to the transaction mode, invoke the operation **begin** on the MBean **ConfigurationAdminMBean**. To commit the accumulated changes, invoke the operation **commit** on the **MBean ConfigurationAdminMBean**. To discard the accumulated changes, invoke the operation **rollback** on the **MBean ConfigurationAdminMBean**.

To release the lock created when the domain was opened, invoke the operation **closeDomain** on the MBean **DomainServiceMBean**.

Switching Domain Configuration Mode

The domain configuration mode specifies how configuration updates are propagated to servers in the domain.

If configuration updates are propagated to all servers in the domain as the changes are done, the domain configuration is online.

If updates are done 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 have them applied the next time a server is restarted. This is used for example when doing a rolling upgrade of an installation.

To switch the domain configuration mode from online to offline, or from offline to online, invoke the operation **setOffline** on the MBean **DomainServiceMBean**.

Mapping Custom Server Names to Names Required by Service Broker

To operate properly, Service Broker imposes the following requirements on naming servers in the Signaling Domain and Processing Domain:

- Names of Signaling Servers must follow the pattern "ssu_<server-number>". For example, the following names are valid: ssu_1, ssu_2, ssu_3.
- Names of Processing Servers must follow the pattern "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 custom server names to names that follow the pattern required by Service Broker. You can perform this mapping using **ServersMBean** and **ServerMBean**.

For more information, see "Mapping Custom Server Names to Service Broker Server Names" in *Oracle Communications Service Broker Configuration Guide*.

Setting a Service Broker Domain Name

After you created a domain, in addition to the name that you assigned to this domain during its creation, you must assign a Service Broker domain name to it.

For more information, see "Setting a Service Broker Domain Name" in *Oracle Communications Service Broker Configuration Guide*.

Adding a Server to a Domain Configuration

Each Signaling Server and Processing Server has a set of server-unique settings that identifies the server in the domain. Table 6–1 describes the server settings.

Table 6–1 Server Settings

Property	Description
name	The name of the server. The name must be unique across all domains. You use this name when you start the server. Format: alpha-numeric characters. Case-sensitive. Do not use white space in the name.
host	The host name or IP-address of the machine where the server will run. Format: alpha-numeric. IP-address format or DNS name format.
port	General purpose IP port to use for traffic to the server. Format: numeric.
adminPort	The IP port to use for propagating configuration updates to the server. Format: numeric

Table 6–1 (Cont.) Server Settings

Property	Description
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. Format: numeric.

To add a server to a domain configuration, invoke the operation **addManagedServer** on the MBean **DomainServiceMBean**. Provide the server settings as parameters to the operation.

When adding a new server to your deployment, the server software needs to be installed, it needs to be added to the domain configuration, and it needs to be configured. For a description of this procedure, see Chapter 7, "Adding and Removing Processing Servers and Signaling Servers".

Removing a Server from a Domain Configuration

To remove a server from a domain configuration, invoke the operation **removeManagedServer** on the MBean **DomainServiceMBean**.

When removing a server from your deployment, the server software needs to be removed, it needs to be removed from the domain configuration, and server-specific settings needs to be removed. For a description of this procedure, see Chapter 7, "Adding and Removing Processing Servers and Signaling Servers".

Managing Domain Properties

Each domain configuration has a set of properties. The properties are name-value pairs. See Table 6–2.

The domain properties are set when the domain configuration is created and they can be edited.

Table 6–2 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 Chapter 1, "Configuring Security".

Table 6–2 (Cont.) Domain Properties

Name	Value
axia.admin.ssl	Specifies if SSL is enabled or disabled for the administration port. Set this property to: ■ **true** to enable security for the administration port. ■ **false** to disable security for the administration port. The administration port is used for propagating configuration updates and to deploy OSGi bundles to servers. Only valid if **axia.ssl** is set to **true**. See Chapter 1, "Configuring Security".
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 domain property, invoke the operation **setDomainProperty** on the MBean **DomainServiceMBean**.

Provide the new property name-value pair to change as parameters to the operation. See Table 6–2.

DomainServiceMBean

JAR File

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

where *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. See "Adding a 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 of the server. Set it to:
  - **ocsb-pn** to create a Processing Domain.
  - **ocsb-ssu** to create a Signaling Domain.
  - **ocsb-basic** to create a combined Domain.
- **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. See "Adding a Server to a Domain Configuration".

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 6–2.

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. See Table 6–1

void renameDomainProperty(String oldName, String newName, String value)

Renames a domain property. See Table 6–2.

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 6–2.

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

where *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 chnages 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.

7 Adding and Removing Processing Servers and Signaling Servers

This chapter describes how to add servers to and remove servers from your Oracle Communications Service Broker deployment:

- Adding a Processing Server or a Signaling Server
- Removing a Processing Server or a Signaling Server

Adding a Processing Server or a Signaling Server

To add a Processing Server or a Signaling Server:

1. Install Service Broker as described in *Oracle Communications Service Broker Installation Guide*.
2. Add the server to the domain configuration. See "DomainServiceMBean" in Chapter 6, "Managing Domains".
3. If the installation is using well-known addresses to join the Processing Domain and the Signaling Domain, add the name, IP-address and port to the Coherence configuration. See section "Joining Domains Using Well-Known Addresses" in Chapter 8, "Managing Clusters".
4. Configure any server-specific configuration settings.
5. Configure security for the server. See Chapter 1, "Configuring Security".
6. Start the Processing Server or Signaling Server. See Chapter 2, "Starting and Stopping Processing Servers and Signaling Servers".

Removing a Processing Server or a Signaling Server

To remove a Processing Server or a Signaling Server:

1. Stop the Processing Server or Signaling Server. See Chapter 2, "Starting and Stopping Processing Servers and Signaling Servers".
2. Use the Administration Console or use scripts to remove all configuration entries that are specific for the server that is removed.
3. If the installation is using well-known addresses to join 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 join domains, perform the following on all domains:

   a. Identify which **UniCastAddress** MBean that corresponds to the server you are removing by checking the attribute **ServerName** for the MBean.
   b. Invoke the operation **removeUnicastAddress** on the MBean **UnicastAddressesMBean**.

4. Un-install the software from the physical server using Oracle Universal Installer. See *Oracle Communications Service Broker Installation Guide* for information on de-installation.

8 Managing Clusters

This chapter describes how to manage your cluster using Coherence:

Understanding Clustering

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

There are two ways to join the domains:

- Using well-known addresses, where the servers in one domain are aware of the IP-addresses, or host-names, of all servers in all other domains.
- Using IP-multicast. This is the default setting when creating the domain using scripts.

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. See section *Network* in *Oracle Coherence Developer's Guide Release 3.5*.

Oracle Coherence is used for clustering. Some of the Configuration MBeans are derived from the Coherence XML configuration. For details about the XML elements, see *Operational Configuration elements* in *Oracle Coherence Developer's Guide Release 3.5*.

Joining Domains Using Well-Known Addresses

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

1. Set the attribute **UseWellKnownAddresses** on the MBean CoherenceConfigTypeMBean.

   Set it to **true**.

2. Invoke the operation **addUnicastAddress** on the MBean **UnicastAddressesMBean**.

   A new Mbean that represents the address of a remote server is added to the MBean structure under **UniCastAddressesMBean**.

3. Set the attribute **serverName** on the MBean **UnicastAddressMBean**.

   Set it to 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.

   Note: The name must be unique across all domains.

   ---

4. Set the attribute **address** on the MBean **SocketAddressTypeMBean**.

   Set it to the host-name or IP-address the server uses for clustering.

   It 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**.

   Set it to the port the server uses for clustering.

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

Example:

You have the servers **pn_1** and **pn_2** defined in the Processing Domain and the servers **ssu_1** and **ssu_2** defined in the Signaling Domain.

You define the following both in the Processing Domain and in the Signaling Domain:

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

  In the sibling **SocketAddressTypeMBean** you set the attribute **address** to the IP-address the server **ssu_1** will use for clustering and the attribute **Port** to the port the server **ssu_1** will use for clustering.

- A **UnicastAddressMBean** with the attribute **serverName** set to **ssu_2**.

  In the sibling **SocketAddressTypeMBean** you set the attribute **address** 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 attribute **serverName** set to **pn_1**.

  In the sibling **SocketAddressTypeMBean** you set the attribute **address** to the IP-address the server **pn_1** will use for clustering and the attribute **Port** to the port the server **pn_1** will use for clustering.

- A **UnicastAddressMBean** with the attribute **serverName** set to **pn_2**.

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

Joining Domains Using IP-Multicast Addresses

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

1. Set the attribute **UseWellKnownAddresses** on the MBean **CoherenceConfigTypeMBean**.

   Set it to **false**.

2. Set the attribute **address** on the MBean **SocketAddressTypeMBean**.

   Set it to the multicast IP address that a socket listens on or publishes to.

   It corresponds to the Coherence XML configuration element **<address>** with the parent-element hierarchy **<multicast-listener>**.

3. Set the attribute **port** on the MBean **SocketAddressTypeMBean**.

   Set it to the port that the socket listens or publishes on.

   It corresponds to the Coherence XML configuration element **<port>** with the parent-element hierarchy **<multicast-listener>**.

4. Set the attribute **Ttl** on the MBean **SocketAddressTypeMBean**.

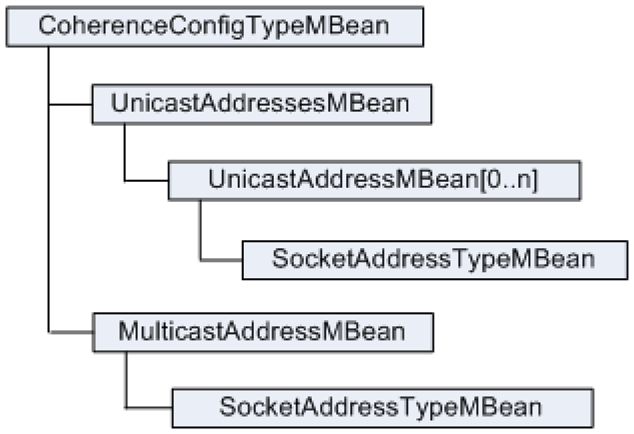
   Specifies the time-to-live setting for the multicast.

   It corresponds to the Coherence XML configuration element **<ttl>** with the parent-element hierarchy **<multicast-listener>**.

Coherence Configuration MBeans

There are a set of MBeans that expose attributes and operations for configuring Coherence through JMX, shown in Figure 8–1.

Figure 8–1 Coherence Configuration MBean Hierarchy



The following sections provide reference information for the Coherence configuration MBeans.

CoherenceConfigTypeMBean

JAR File

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

where *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 "Understanding 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

where *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 3.5*.

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

where *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

where *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 3.5*.
- 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 3.5*

Operations

None

UnicastAddressesMBean

JAR File

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

where *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

where *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

where *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**.

9 Upgrading and Patching

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 may 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 continues 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. Backup your Processing Servers, Signaling Servers, and domain configurations prior to performing an upgrade, see Chapter 10, "Maintaining Oracle Communications Service Broker".
2. If the patch is for a bundle, apply the patch on the domain configuration. See "Applying Patches for Bundles".
3. If the patch is for a server, apply the patch on each server. See "Applying Patches for Servers".

4. On each server:

   Perform a controlled shutdown of the server and start it again. See Chapter 2, "Starting and Stopping Processing Servers and Signaling Servers".

Applying Patches for Bundles

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

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

   where *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* as when you installed Oracle Communications Service Broker.

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

   Oracle_home/**patch**
6. Set the domain configuration to offline by setting the attribute **OffLine** on the **DomainServiceMBean** to **true**. See Chapter 6, "Managing Domains".
7. Uninstall the bundle(s) to update using the operation **uninstallBundle** in **DeploymentServiceMBean** method. See "DeploymentServiceMBean".
8. 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. Login to the computer where the server to be patched is installed.
2. Unzip the ZIP file for to the directory *patch_directory*.

   where *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* as 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.

Deployment Service MBean

The Deployment service MBean exposes operations for installation and deployment of OSGi bundles.

The following sections provide reference information for the Deployment service MBean.

DeploymentServiceMBean

The DeploymentServiceMBean provides operations for OSGi bundle management.

JAR File

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

where *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.

10 Maintaining Oracle Communications Service Broker

This chapter describes how you maintain your deployment and to perform housekeeping. For details, see

Backing Up Your Installation

Back-up can be done on a set of different levels:

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

Backing Up a Processing Server or Signaling Server

To back-up a Processing Server or Signaling Server, back-up everything under the following directory:

Oracle_home/**axia/managed_server**

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

You should do a backup immediately after you installed, upgraded, or patched your Processing Server or Signaling Server.

If you want to backup any log files, see "Archiving and Cleaning Up Log Files".

Backing Up an Administration Client

When backing up an administration client, the following is backed up:

- The Stand-alone Administration Console
- The Web administration Console server

- The Scripting Engine

To backup an administration client, back-up everything under the following directory:

*Oracle_home***/axia/admin_console**

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

You should backup immediately after you install, upgrade, or patch your administration client.

If you want to backup any log files, see "Archiving and Cleaning Up Log Files".

Backing Up a Domain Configuration

To backup your domain configuration, backup everything in your domain configuration directory. This directory was defined when you created the domain.

You should perform backups on a regular basis and always immediately after you:

- Update or change any configuration.
- Add or remove a Processing Server or Signaling Server from your installation.
- Upgrade a Processing Server or Signaling Server.
- Patch a Processing Server, Signaling Server, or an administration client.

Backing Up an Oracle Home

To backup your full Oracle Home, backup 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***/axia/managed_server**

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

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

*Oracle_home***/axia/admin_console**

Where *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***/axia**

Where *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 the Log4J documentation at:

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

11 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 your 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 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 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 which name is the same as the parameter name defined by the MBean.

Table 11–1 describes the syntax of the script file.

Table 11–1 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 not 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 is deployed. The attribute **port** corresponds to the JMX Registry port defined for the Processing Server and Signaling Server. The JMX Registry port for Processing Servers and a 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 JavaDoc for the MBeans 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]. For complex data types, see "Representation of Complex Data Structures".

Table 11–1 (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:

Example 11–1 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 11–1 Getting an MBean Attribute

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

Example:

Example 11–2 describes how to set the attribute **UseWellKnownAddress** on the MBean **CoherenceConfigTypeMBean**. The MBean is retrieved by 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 11–2 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 11–3 describes how to invoke the operation with the signature **void openDomain(java.lang.String domainPath)** with **domainPath** set to **/usr/local/sb/domain**.

Example 11–3 Invoking an MBean Operation

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

Using Variables

Variables can be used in scripts. The notation when using a variable is:

${*variable_name***}**

Variables are set using system properties.

You set system properties by defining the variable using the Java **-D** flag. Set it in the environment variable **AXIA_OPTS**. The syntax is

SET AXIA_OPTS=-D*variable_name***=***variable_value*

For example, to set the variable **domain.path** to **/domains/ocsb-basic-fs** set the environment variable **AXIA_OPTS** using the command:

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

The value of **domain.path** is used in the script when the variable is referenced. Below is an example of how to reference it:

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

If the variable is not defined, the Scripting engine prompts for a value of the variable. 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 as illustrated below:

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

Representation of Complex Data Structures

Complex data structures can be used as in-parameters to operations. When referring to elements of complex data structures in Java, the elements are normally addresses 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:

```
...
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 11–4 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 11–4 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***/axia/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.

12
Life Cycle of Processing Servers and Signaling Servers

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

- Life Cycle
- Life Cycle Management MBeans
- ManagementAgentMBean

Life Cycle

Oracle Communications Service Broker has life cycle states and transitions as illustrated in Figure 12–1. The life cycle is started when Service Broker is started. 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 12–1 States and State Transitions

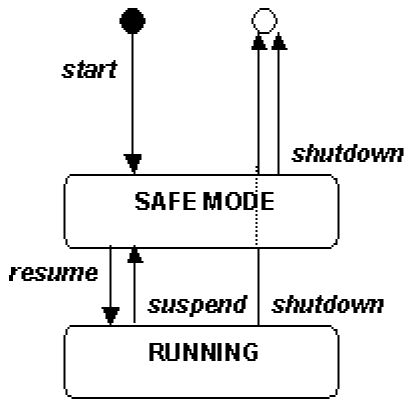


Table 12–1 describes the OSGi states and state transitions, and gives information on the corresponding OSGi start levels.

Table 12–1 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.
SHUTDOWN	*start*	This state transition is triggered by the server start script. Safe services are started. See Appendix A, "System Administrator's Reference". 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 start-up. That is, before entering state SAFE MODE from state initial. During this transition, the running OSGi bundles are shutdown. 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 transitions 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 MBeans.

ManagementAgentMBean

ManagementAgentMBean enables you to perform life cycle 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.

13 Monitoring Service Broker

The following sections explain how you can 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 statuses
- 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 machine 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.

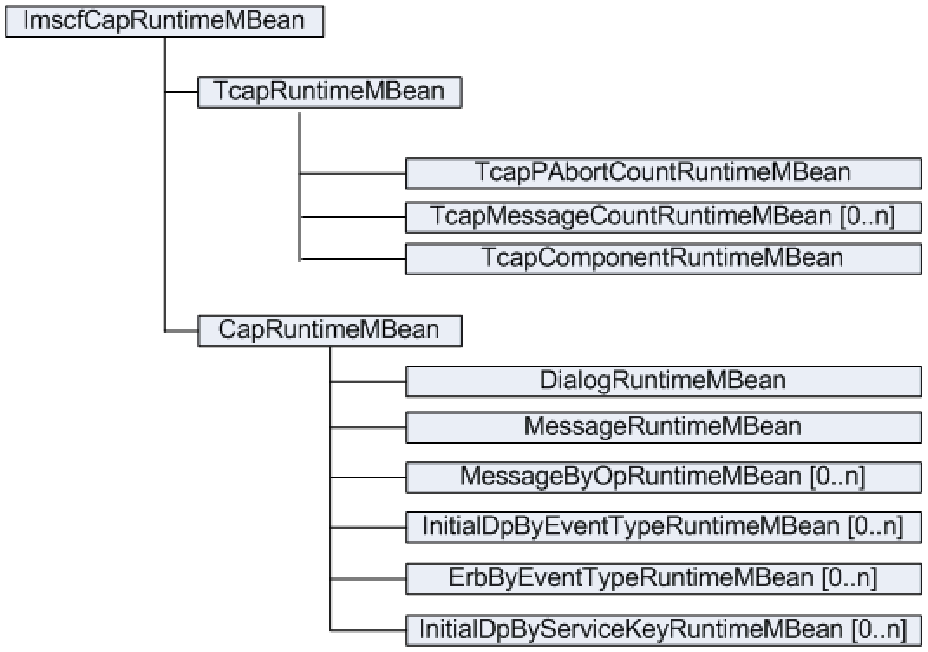
Service Broker Runtime MBeans Organization

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 13–1 shows an example of the Runtime MBean hierarchy for the IM-SCF CAP phase 1 Interworking Module:

Figure 13–1 Example of the IM-SCF CAP Runtime MBean Hierarchy


ImscfCapRuntimeMBean
TcapRuntimeMBean
TcapPAbortCountRuntimeMBean
TcapMessageCountRuntimeMBean [0..n]
TcapComponentRuntimeMBean
CapRuntimeMBean
DialogRuntimeMBean
MessageRuntimeMBean
MessageByOpRuntimeMBean [0..n]
InitialDpByEventTypeRuntimeMBean [0..n]
ErbByEventTypeRuntimeMBean [0..n]
InitialDpByServiceKeyRuntimeMBean [0..n]

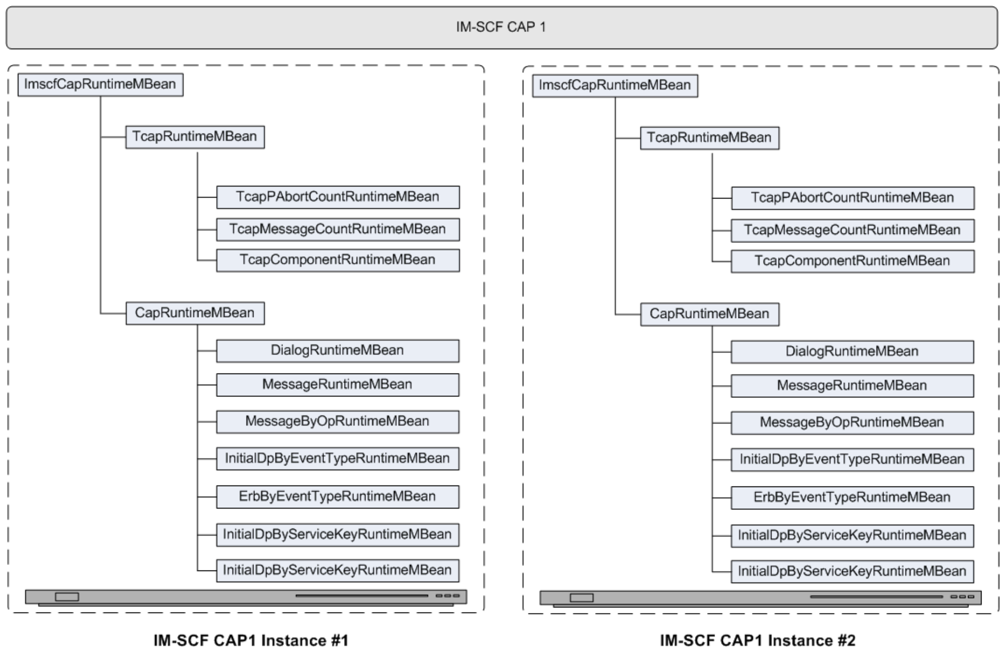
Each Runtime MBean provides reference to other Runtime MBeans below in the hierarchy. Therefore, a JMX client can access a Runtime MBean by either directly looking it up in the MBean Server or by browsing down through the Runtime MBean hierarchy.

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 13–2 shows an example of how IM-SCF instantiates Runtime MBeans on each of the servers on which IM-SCF is installed.

Figure 13–2 IM-SCF CAP1 Instances


IM-SCF CAP 1
ImscfCapRuntimeMBean
TcapRuntimeMBean
TcapPAbortCountRuntimeMBean
TcapMessageCountRuntimeMBean
TcapComponentRuntimeMBean
CapRuntimeMBean
DialogRuntimeMBean
MessageRuntimeMBean
MessageByOpRuntimeMBean
InitialDpByEventTypeRuntimeMBean
ErbByEventTypeRuntimeMBean
InitialDpByServiceKeyRuntimeMBean
InitialDpByServiceKeyRuntimeMBean
IM-SCF CAP1 Instance #1
IM-SCF CAP1 Instance #2

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, it also destructs its Runtime MBeans.

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 13–1 describes the key properties that Service Broker encodes in its Runtime MBean object names.

Table 13–1 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 13–1 (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 an 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 a 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 a number of times that a specific event occurred by the end of a previous time interval ■ CurrentGeneral, when you want to get a 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), 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.
- Lookup 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 Runtime MBeans Javadoc*.

Service Broker Measurements

This section describe various aspects of Service Broker measurements.

Counters, Gauges, TPSs and Statuses

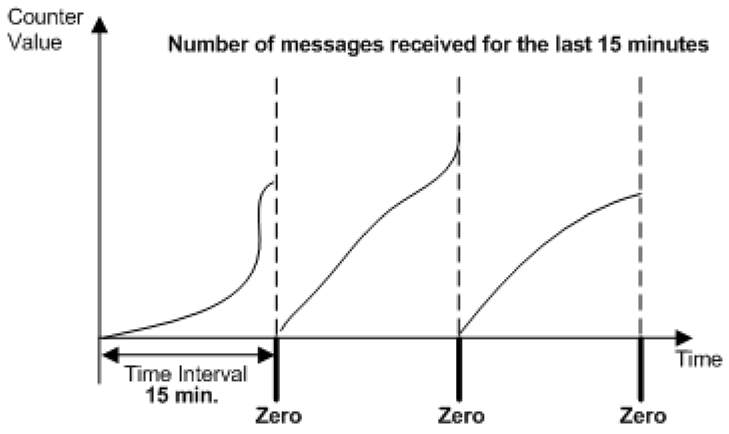
Service Broker Runtime MBeans provide the following types of attributes:

Counters

Counters store the number of times a particular event has occurred in the last time interval. For example, a counter can provide the number of messages received in the last 15 minutes 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 13–3 shows how a counter increases and zeroes periodically in the end of each time interval.

Figure 13–3 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 a number of times that a specific event occurred from the beginning of the time interval till the moment when 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.

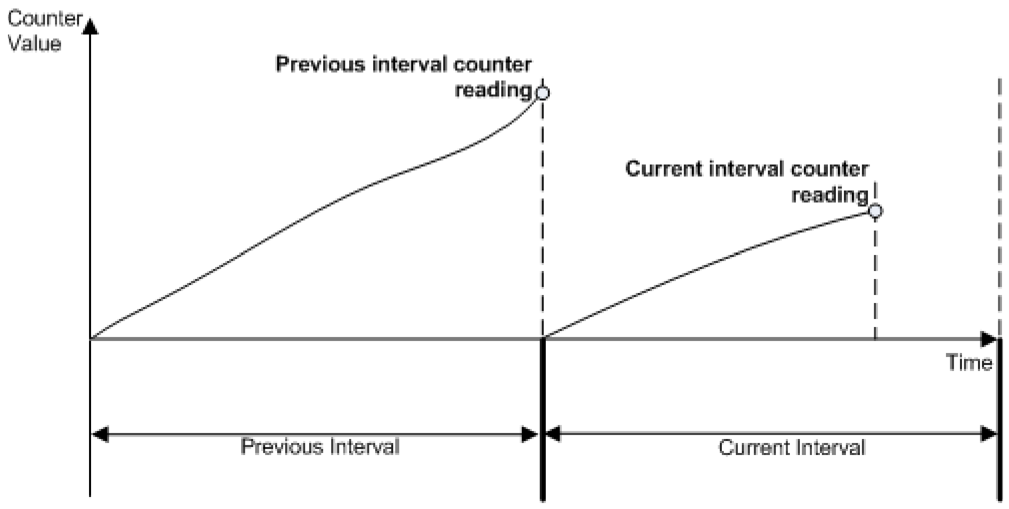
- Previous interval counters

  This type of counters provides a 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 13–4 shows an example of a current interval counter compared to a previous interval counter.

Figure 13–4 Current Interval Counter


Counter Value
Previous interval counter reading
Current interval counter reading
Time
Previous Interval
Current Interval

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.

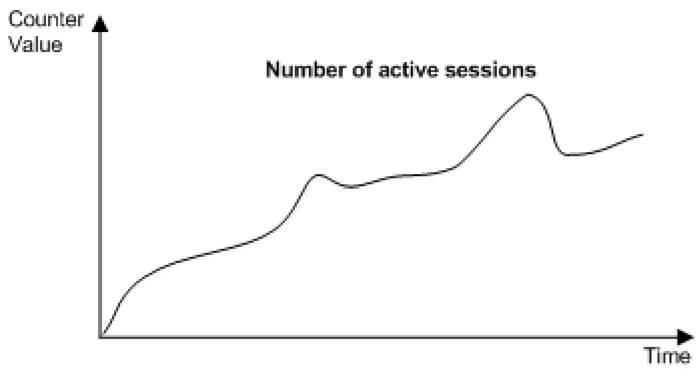
Names of TPS attributes have the **count** prefix. However, as opposed to regular counter attributes, TPS counters are available in MBean instances whose CountingMethod property is set to ShortInterval.

Gauges

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

Figure 13–5 shows how a gauge's value changes over time.

Figure 13–5 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 13–2 explains how reaching an upper threshold or a lower threshold causes reporting and clearing of a notification.

Table 13–2 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 13–3 explains how reaching an upper threshold or a lower threshold causes reporting and clearing of a notification.

Table 13–3 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 13–4 explains how entering a specified state and exiting a state causes reporting and clearing of a notification.

Table 13–4 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 13–5 describes notification classes that can be defined in the Type field.

Table 13–5 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

  This specifies the time when the event which triggered the notification, occurred
- UserData

  This 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 tree view of MBeans is displayed.
3. On 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 on 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 asses 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 will 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 13–6 describes attributes that enable you to monitor the TCAP interface.

Table 13–6 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	TcapPAbortCountRuntime MBean
BadlyFormattedTransaction PortionCount		Counter	TcapPAbortCountRuntime MBean

Table 13–6 (Cont.) TCAP Interface Monitoring Attributes

Attribute	Description	Type	MBean
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 13–6 (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 13–6 (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 13–7 describes attributes that enable you to monitor the IN interface.

Table 13–7 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
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 13–8 describes attributes that enable you to monitor the Diameter interface

Table 13–8 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
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

Table 13–8 (Cont.) Diameter Monitoring Attributes

Attribute	Description	Type	MBean
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 13–9 describes attributes that enable you to monitor the SIP interface

Table 13–9 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
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 13–10 describes attributes that enable you to monitor the Service Broker Processing Tier.

Table 13–10 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 13–11 describes attributes that enable you to monitor SS7 SSU.

Table 13–11 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
Status	Specifies a point code status. Possible values for TDM: ■ 1 - Remote SCCP is available ■ 2 - Remote SCCP is unavailable, reason unknown ■ 3 - Remote SCCP is unequipped ■ 4 - Remote SCCP is inaccessible ■ 5 - Remote SCCP is congested Possible values for SIGTRAN: ■ 0 - Remote SCCP is accessible ■ 1 - Remote SCCP is inaccessible	Status	SsuRemotePointCode RuntimeMBean

Monitoring SIP SSU

Table 13–12 describes attributes that enable you to monitor the SIP SSU.

Table 13–12 SIP SSU Monitoring Attributes

Attribute	Description	Type	MBean
Status	Specifies the status of a SIP network entity. Possible values: ■ Active ■ Inactive	Status	NetworkEntityRuntime MBean

Monitoring the Orchestration Engine

Table 13–13 describes attributes that enable you to monitor the OE.

Table 13–13 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
ExecutionCount	Returns the number of times that the OE triggered the specific OLP in the last measurement period	Counter	OlpRuntimeMBean
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 13–14 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 13–14 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
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 13–15 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 13–15 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
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

Table 13–15 (Cont.) IM-SSF Monitoring Attributes

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

Monitoring IM-ASF

Table 13–16 describes attributes that enable you to monitor IM-ASF.

Table 13–16 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 13–17 describes attributes that enable you to monitor R-IM-ASF.

Table 13–17 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 13–18 describes attributes that enable you to monitor the root ImocfRuntimeMBean. For more information on monitoring the Diameter interface, see "Monitoring the Diameter Interface".

Table 13–18 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
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 R-IM-OCF

You can monitor the following:

- Root ImocfRuntimeMBean
- Diameter interface

Table 13–19 describes attributes that enable you to monitor the root RimocfRuntimeMBean. For more information on monitoring the Diameter interface, see "Monitoring the Diameter Interface".

Table 13–19 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

Table 13–19 (Cont.) R-IM-OCF Monitoring Attributes

Attribute	Description	Type	MBean
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-PSX

You can monitor the following interfaces of IM-PSX:

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

Table 13–20 describes attributes that enable you to monitor the MAP interface. For more information on monitoring the TCAP interface, see "Monitoring the TCAP Interface".

Table 13–20 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 MessageByOpRuntimeMB ean

Monitoring SM-PME

Table 13–21 describes attributes that enable you to monitor the IN interface

Table 13–21 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

14 Viewing Service Broker SDRs

The following sections explain how you can monitor Oracle Communications Service Broker using Service Broker Service Description Records (SDRs):

- Understanding Service Broker Service Description Records
- Configuring SDR Logging
- Service Description Record Format

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>
```

Where:

- <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 files 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 the 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.log4j config,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.log4j config,version=1.0.0.0,name0=log4jConfig,name1=configuration[0],name2=logg er[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 14–1 describes the parameters available in each SDR.

Table 14–1 Service Broker SDR Fields

Field	Description
ModuleType	Specifies the type of Service Broker module that generated the SDR. This value of 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.

Table 14–1 (Cont.) Service Broker SDR Fields

Field	Description
CallingPartyNumber	Specifies the calling party number
AdditionalCallingPartyNumber	Specifies the calling party number. Usually this number is identical to the value of the CallingPartyNumber field.
CalledPartyNumber	Specifies the called party number
OPR	Specifies the Orchestration Profile Receiver that the OE used to obtain the orchestration profile. For more information about the types of OPRs, see "About Orchestration Profile Receivers" in the *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 was 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>, where: ▪ <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.

An example of an SDR is provided below:

```
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 into the SDR. The data consists of a range of tags, which vary according to the IM type.

Table 14–2 describes the tags generated by the IM-OCF.

Table 14–2 IM-Generated Tags

IM Type	Tags	Description
IMOCF	sid=35263; recum=3; nonChargeDuration=25	sid: session id reqnum: the number of the last successful CCR request sent to an OCF nonchargeDuration: 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.

15

Preventing System Overload

The following sections explain how you can protect Oracle Communications Service Broker from overload:

- Understanding Overload Protection
- Configuring Overload Protection

Understanding Overload Protection

In some cases, such as traffic peaks or unexpected failure of network entities, the load on Service Broker modules may significantly increase. 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, and enables you to define criteria for overload detection and configure how Service Broker behaves if system overload occurs.

To activate overload protection you have to go through the following steps:

1. Identify key overload indicators

   You can choose any out 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 will trigger 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 will later define the gauges and counters as key overload indicators, on step 3.

2. Define thresholds for the identified indicators

   For each key overload indicator (that is gauge or counter) that you identified in the previous step, you need to define an upper threshold. When a key overload indicator crosses its 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, any of the key overload indicators crosses its threshold), it continues handling active sessions but stops accepting new session. 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 use 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 "**Managing the Service Broker Processing Tier**" in *Oracle Communications Service Broker Configuration Guide.*

A

System Administrator's Reference

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

Details for the Administration Clients

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

Directory Structure and Contents for an Administration Client

All administration clients are installed under the directory:

Linux and Solaris: *Oracle_home***/axia/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/axia

Directory	Description
admin_console	Top-level directory for all administration clients. It contains start-scripts for: ▪ The Stand-alone Administration Console ▪ The Web Administration Console server ▪ The Scripting Engine ▪ The Domain Web server It 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/config	Contains configuration data.
admin_console/modules	Contains all OSGi bundles for the administration clients, the Processing Server, and the Signaling Server.
admin_console/osgi	Contains platform-specific binaries and configuration files.
admin_console/properties	It contains property files used by the start-scripts for: ▪ The Stand-alone Administration Console ▪ The Web Administration Console server ▪ The Scripting Engine ▪ The Domain Web server
admin_console/workspace	Contains meta-data for the 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. It defines the environment variables: ▪ **OSGI_CONFIG** ▪ **BOOT_CONFIG** ▪ **BOOT_OPTS** **script.sh** calls **common.sh**. For more information, see Chapter 11, "Using Scripts for Configuration and Management".

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

Script	Description
start.sh	Starts the Stand-alone Administration Console. It defines the environment variables: ■ **OSGI_CONFIG** ■ **BOOT_CONFIG** ■ **BOOT_OPTS** **start.sh** calls **common.sh**.
web.sh	Starts the Web Administration Console server. It defines the environment variables: ■ **OSGI_CONFIG** ■ **BOOT_CONFIG** ■ **BOOT_OPTS** **web.sh** calls **common.sh**. For information on how to use the script, see Chapter 4, "Starting and Stopping Web Administration Console Servers".
host.sh	Starts the Domain Web server. It defines the environment variables: ■ **OSGI_CONFIG** ■ **BOOT_CONFIG** ■ **BOOT_OPTS** **host.sh** calls **common.sh**. For information on how to use the script, see Chapter 5, "Starting and Stopping Domain Web Servers".
common.sh	Starts the Stand-alone 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. It defines the environment variables: ■ **JAVA_CMD** ■ **JAVA_OPTS** Defines the system property **boot.properties**.

Property Files for the Administration Clients

Table A–3 gives information property files in:

Oracle_home/**admin_console/properties**

Table A–3 Property files used by the Administration Clients

Property File	Description
common.properties	Defines properties common for the: ▪ The Stand-alone Administration Console ▪ The Web Administration Console server ▪ The Scripting Engine ▪ The Domain Web server The properties specified are: ▪ **axia.console.log4j.server.port** ▪ **axia.ssl** ▪ **javax.net.ssl.keyStore** ▪ **javax.net.ssl.trustStore** ▪ **log4j.configuration** See Table A–8.
hosting.properties	Defines properties for the Domain Web server. The properties specified are: ▪ **axia.platform** ▪ **org.eclipse.equinox.http.jetty.http.port** See Table A–8.
script.properties	Defines properties for the Domain Web server. The properties specified are: ▪ **axia.platform** ▪ **org.eclipse.equinox.http.jetty.http.port** See Table A–8.
standalone.properties	Defines properties for the Stand-alone Administration Console. The properties specified are: ▪ **axia.platform** ▪ **org.eclipse.equinox.http.jetty.http.enabled** ▪ **org.eclipse.equinox.http.jetty.https.enabled** See Table A–8.
web.properties	Defines properties for the Web Administration Console server. The properties specified are: ▪ **axia.platform** ▪ **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** See Table A–8.

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***/axia/managed_server**

Where *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/axia

Directory	Description
managed_server	Top-level directory for a Processing Server and a Signaling Server. It contains start-scripts for the Processing Server and the Signaling Server. I contains the property file **server.properties**. It also contains files related to Log4J: ▪ **server.log** file 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 Log4JService 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: ▪ the platform logging service ▪ Log4J ▪ provisioning service The bundles in this directory is the minimal set necessary to initiate the server and load the contents of the domain configuration directory.
managed_server/osgi	This directory, sub-directories, and contents are created as a part of the server start life cycle and mirrors the content of the domain configuration file. Synchronized with the domain configuration.
managed_server/osgi	Contains binaries for the SS7 stacks for TDM and Sigtran.

Start Scripts for Processing Servers and Signaling Servers

Table A–5 give 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 Stand-alone Administration Console. Defines the environment variables: ▪ **JAVA_CMD** ▪ **JAVA_OPTS** ▪ **AXIA_MEM_OPTS** Defines properties for the servers. The properties specified are: ▪ **provisioning.spid** ▪ **org.osgi.ip.URL** ▪ **boot.properties** See Table A–6. For a syntax description, see Chapter 2, "Starting and Stopping Processing Servers and Signaling Servers".

Property Files for 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** See Table A–8.

Environment Variables

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

Table A–7 Environment variables

Variable	Description
OSGI_CONFIG	Defines the OSGi configuration and bundles. Must be set to: ▪ **osgi/script** in **script.sh** ▪ **osgi/hosting** in **host.sh** ▪ **osgi/standalone** in **start.sh** (Stand-alone Administration Console) ▪ **osgi/web** in **web.sh**
BOOT_CONFIG	Defines which property file to use. See Table A–3.

Table A–7 (Cont.) Environment variables

Variable	Description
BOOT_OPTS	Defines system properties used at boot time. See Table A–8.
JAVA_CMD	Defines the JRE to use.
AXIA_OPTS	Defines any additional Java options to use.
AXIA_MEM_OPTS	Defines the Java heap size. Sets the flag **-Xmx** when starting the JRE.
JAVA_OPTS	Defines the system property **boot.properties** and aggregates the following environment variables: ▪ **BOOT_OPTS** ▪ **AXIA_OPTS** ▪ **AXIA_MEM_OPTS**

System Properties

Table A–8 describes the system properties defined for Oracle Communications Service Broker.

Table A–8 Description of System Properties

System Property	Description
axia.console.log4j.server.port	The port to use for the logging service. Set in **common.properties**
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.
provisioning.spid	Defines the name of the Processing Server or Signaling Server. Provided as an argument to the command. Set in **start.sh** (Processing Server and Signaling Server).
org.osgi.ip.URL	Defines the URL of the domain configuration file. HTTP and FILE can be used as the protocol part. Set in **start.sh** (Processing Server and Signaling Server).
org.eclipse.equinox.http.jetty.http.port	Defines the port the Web server listens to. 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.
axia.domain.path	Defines the URL to the domain configuration. Provided as an argument to the command. Set in **web.sh**.

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

System Property	Description
axia.hosting.domain.path	Defines the URL to the domain configuration. Provided as an argument to the command. Set in **host.sh**.
axia.console.script	Defines the script to execute. Provided as an argument to the command. Set in **script.sh**.
axia.ssl	Specifies if SSL is enabled or disabled. Set this property to: ■ **true** to enable security. ■ **false** to disable security. Set in **common.properties**.
javax.net.ssl.keyStore	The filename of the keystore to use for processing servers, signaling servers and the 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 filename of the truststore to use for processing servers, signaling servers and the 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.
axia.platform	It defines the start mode. This must not be changed. Must be set to: ■ **server** in **server.properties**. ■ **standalone** in **standalone.properties**. ■ **web** in **web.properties**. ■ **script** in **script.properties**. ■ **hosting** in **hosting.properties**. Set in: ■ **start.sh** (Stand-alone Administration Console) ■ **server.properties** ■ **web.properties** ■ **script.properties** ■ **hosting.properties**

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

System Property	Description
org.eclipse.equinox.http.jetty.http.enabled	Specifies if HTTP is used or not. Set this property to: ■ **true** to use HTTP. ■ **false** to not use HTTP. Set in: ■ **script.properties** ■ **standalone.properties** ■ **web.properties** Must always the Boolean NOT of the property **org.eclipse.equinox.http.jetty.https.enabled** in **web.properties**. Must always be set to **false** in **script.properties** and standalone.properties.
org.eclipse.equinox.http.jetty.https.enabled	Specifies if HTTPS is used or not. Set this property to: ■ **true** to use HTTPS. ■ **false** to not use HTTPS. Set in: ■ **script.properties** ■ **standalone.properties** ■ **web.properties** Must always the Boolean NOT of the property **org.eclipse.equinox.http.jetty.http.enabled** in **web.properties**. Must always be set to **false** in **script.properties** and standalone.properties.
axia.basic.auth	Specifies if HTTP basic authentication is used or not 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**.
org.eclipse.equinox.http.jetty.https.port	Specifies which port to use for HTTPS connections. Set in **web.properties**.
org.eclipse.equinox.http.jetty.ssl.keystore	Specifies the keystore to use for the HTTPS connection between the Web Administration Console and the Web Administration Console server. This property i not set by default. If not set, the same keystore as defined in the property **javax.net.ssl.keyStore** is used. Set in **web.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**.

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

System Property	Description
boot.properties	Specifies which property file to use. Do not change this setting. Set in: ■ **common.sh** ■ **start.sh** (Processing Server and Signaling Server)

Directory Contents and Structure for a Domain Configuration

Table A–9 give information about the directory structure and contents of a domain configuration.

Table A–9 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 a Signaling Servers in the domain. ■ **admin_lock.dat** Lock file used to ensure exclusive write-access to the domain configuration.
Domain_home/**modules**	Contains binaries and configuration data for Processing Servers and Signaling Servers in the domain.

Directory Structure and Contents for JDKs

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

These files are located in under the directory:

Linux and Solaris: *Oracle_home*/**axia**

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

Table A–10 describes the directory structure and the contents of the directory structure.

Table A–10 Directory Structure for JDKs Relative to Oracle_home/axia

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.

Table A–10 (Cont.) Directory Structure for JDKs Relative to Oracle_home/axia

Directory	Description
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/**axia**

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

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

Table A–11 Directory Structure for Oracle Universal Installer Relative to Oracle_home/axia.

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–12 lists these services.

Table A–12 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