Oracle® Communications Services Gatekeeper

System Administrator's Guide

Release 6.0

E50758-03

November 2015

Oracle Communications Services Gatekeeper System Administrator's Guide, Release 6.0

E50758-03

Contents

3 Starting, Stopping, and Administering Servers

4 Managing, Backing Up, and Restoring Services Gatekeeper

5 Securing Services Gatekeeper

6 Setting Up Geographic Redundancy

7 Configuring Coherence Clusters

8 Managing Users and User Groups

9 Configuring and Using the SNMP Service

10 Generating Statistics for Transaction Licenses

11 Managing and Configuring EDRs, CDRs and Alarms

12 Resolving Policy Deny Codes

13 Managing and Configuring the Trace Service

14 Troubleshooting Your Services Gatekeeper Implementation

15 Using Cloud Monitoring with Multi-tier Services Gatekeeper

Part II Services Gatekeeper Advanced Administration

16 Managing and Configuring SLAs

21 Upgrading and Redeploying Communication Services and Service Interceptors

22 Managing and Configuring the Tier Routing Manager

23 Charging and Integrating Billing

24 Implementing Diameter Ro Online Charging

25 Implementing Diameter Rf Offline Charging

26 Managing and Configuring Native UCP Connections

27 Managing and Configuring Parlay X 2.1 Shortcode Mappings

28 Managing OSA/Parlay Gateway Connections Using Parlay_Access

29 Managing Legacy Application Service Providers

Preface

This guide describes the system administration tasks involved in configuring and managing an Oracle Communications Services Gatekeeper implementation.

Audience

This book is intended for technical personnel who will administer a Services Gatekeeper implementation.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents in the Services Gatekeeper set:

- *Oracle Communications Services Gatekeeper Concepts*
- *Oracle Communications Services Gatekeeper Getting Started Guide*
- *Oracle Communications Services Gatekeeper Alarms Handling Guide*
- *Oracle Communications Services Gatekeeper Application Developer's Guide*
- *Oracle Communications Services Gatekeeper Communication Service Reference Guide*
- *Oracle Communications Services Gatekeeper Multi-tier Installation Guide*
- *Oracle Communications Services Gatekeeper Licensing Guide*
- *Oracle Communications Services Gatekeeper OAuth Guide*
- *Oracle Communications Services Gatekeeper Security Guide*

1

Administration Overview

This chapter provides an overview of the tasks required to configure and administer Oracle Communications Services Gatekeeper.

Services Gatekeeper Configuration Workflow

This section describes an overall workflow for configuring a Services Gatekeeper implementation.

1. The first step is to see the "Services Gatekeeper Administration Tools and Database Maintenance" chapter for an overview of the Services Gatekeeper administration tools.
2. If you do not use the PRM portals, the next step is to set up administrative users for the Administration Console to use, see:
   - Managing Users and User Groups
3. For basic Services Gatekeeper administration tasks see:
   - Starting, Stopping, and Administering Servers
   - Managing, Backing Up, and Restoring Services Gatekeeper
   - Securing Services Gatekeeper
4. For Services Gatekeeper administrations tasks as needed, see:
   - Setting Up Geographic Redundancy
   - Configuring Coherence Clusters
   - Configuring and Using the SNMP Service
   - Managing and Configuring SLAs: This expands on the basic settings used by the PRM portals.
   - Generating Statistics for Transaction Licenses
   - Managing and Configuring EDRs, CDRs and Alarms
   - Resolving Policy Deny Codes
   - Managing and Configuring the Trace Service
   - Upgrading and Redeploying Communication Services and Service Interceptors
5. For Services Gatekeeper administration tasks if you manually administer and use communication services see:
   - Deploying and Administering Communication Services

   - Managing and Configuring the Storage Service
   - Configuring Network Node Heartbeats
   - Configuring and Managing Communication Service Traffic
   - Managing and Configuring the Tier Routing Manager
6. For instructions to configure charging with a Diameter Server, see:
   - Implementing Diameter Ro Online Charging
   - Implementing Diameter Rf Offline Charging
7. To configure Services Gatekeeper to work with Open Services Architecture (OSA)/Parlay Gateways, see:
   - Managing OSA/Parlay Gateway Connections Using Parlay_Access
8. If you use Native UCP connections see:
   - Managing and Configuring Native UCP Connections
9. If you use Parlay X shortcode mappings:
   - Managing and Configuring Parlay X 2.1 Shortcode Mappings

For details on the individual communication services, see *Services Gatekeeper Communication Service Reference Guide*.

After the system is configured, provision service providers and applications, as described in the discussion on creating and maintaining service provider and application account in the *Services Gatekeeper Portal Developer's Guide*.

Part I

Services Gatekeeper Administration

This part provides an overview of Oracle Communications Services Gatekeeper architecture and management, and provides administration information applicable to all Services Gatekeeper implementations.

Part I contains these chapters:

2

Services Gatekeeper Administration Tools and Database Maintenance

This chapter introduces Oracle Communications Services Gatekeeper configuration and administration.

About Services Gatekeeper Installation

The single-tier version of Services Gatekeeper is designed to deploy quickly on a single or clustered self-contained system. When you install the single-tier version of Services Gatekeeper, the resulting system already configured and ready to run. The multi-tier version of Services Gatekeeper includes all the features of a single-tier Services Gatekeeper, but is intended for larger and more complex implementation and requires configuration. You can extend it into a more robust and flexible environment.

For an overview of Services Gatekeeper features and the types of implementations, see “Understanding Services Gatekeeper” in *Services Gatekeeper Concepts*.

About the Oracle WebLogic Platform

Services Gatekeeper is based on Oracle WebLogic Server. Many system-level configuration tasks are the same for both products. This chapter addresses the system-level configuration tasks that are unique to Services Gatekeeper. These tasks relate to network and security configuration and cluster configuration for the engine.

WebLogic Server configuration and other basic configuration tasks such as logging are addressed in the WebLogic Server documentation. This guide refers you to the WebLogic documentation for information where appropriate.

Services Gatekeeper configurations are built using a version of Oracle WebLogic Server 12c. See the Oracle Fusion Middleware 12c website for more information about WebLogic:

`http://docs.oracle.com/middleware/1212/wls/index.html`

Using the Administration Console to Configure Services Gatekeeper

Services Gatekeeper Administration Console is a graphical user interface (GUI) for configuring, managing, and provisioning Services Gatekeeper. The Administration Console is an extension of the WebLogic Server Administration Console.

About Using the Administration Console

Services Gatekeeper runs a WebLogic administration server for configuration which is accessible through the Administration Console. The Administration Console exposes the underlying JMX MBean attributes and operations in GUI form. These MBeans are also available for your to use programmatically.

Run an instance of the Administration Console on every system with an administration server.

The single-tier version of Services Gatekeeper uses a single administration server for the entire implementation. Run one instance of the Administration Console to configure the server.

Multi-tier Services Gatekeeper implementations however, generally run different administration servers for Services Gatekeeper and for the partner relationship management (PRM) portals. You run an Administration Console on each of those systems to administer them.

Starting and Using the Administration Console

At least one network tier server must be started before you can log in to the WebLogic Server Administration Console. Otherwise, the configuration settings do not display in the Administration Console. To see them, log in to WebLogic Server Administration Console again.

Logging in to the Administration Console

To log in to the Administration Console:

1. Open a supported web browser and go to the following URL:

   http://*server_address*:*port***/console**

   Where:

   – *server_address* is the instance you have set up as your administration server, usually the IP address of the system. Use the entry **localhost** when the Administration Console is on the same system as the administration server.

   – *port* is the port to connect to. The default is **7001**.

2. Log with your credentials.

> **Note:** The user name `weblogic` and password `weblogic` are the recommended values for the installation and initial domain configuration stage. For the first Services Gatekeeper session, enter these values to log in.
>
> After you have logged in, create an administrative user using the instructions in "Managing Users and User Groups", and remove the user `weblogic`. For information in removing or changing the password for the WebLogic Server user, see the on-screen help text on the Administration Console user interface.

About the Administration Console

The Administration Console user interface consists of a set of selection panels to the left and to the right, the page or pages used for configuring and monitoring the selected node in the specific domain:

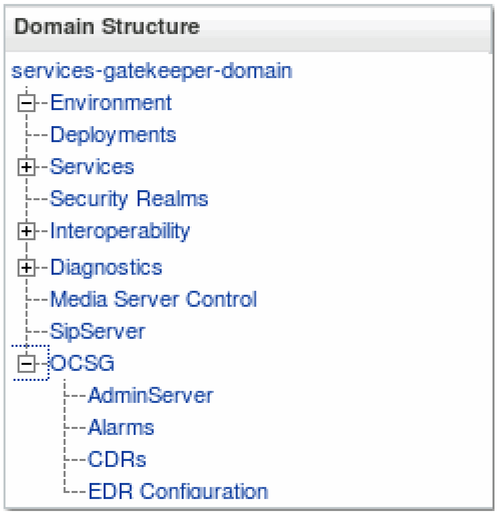
- **Change Center**, the starting point for using the Administration Console to configure the current domain. To prevent other accounts from updating the configuration simultaneously, lock the editable configuration hierarchy for the domain. To do so, click **Lock & Edit**.
- **Domain Structure**, a tree structure you can use to navigate to pages in the Administration Console. See "Navigating the Domain Structure".
- **How do I ...**, with links to online help tasks that are relevant to the current Console page.
- Toolbar, at the top of the Console with important details such as your access name, the connection address, and links.
- Bread Crumb Navigation, a series of links that show the path you have taken through the pages of the Administration Console.
- **System Status**, a panel that reports on the number of information, error, and warning messages that have been logged.

For more information, see the discussion "About the Administration Console" in *Fusion Middleware Understanding Oracle WebLogic Server*.

Navigating the Domain Structure

Figure 2–1 displays the **Domain Structure**, located in left pane of the console. To view the page for a node, select the node in the Domain Structure tree. To expand a node or to collapse the node, click a + (plus) icon or a - (minus) icon in the **Domain Structure**.

Figure 2–1 Domain Structure Setup



```
Domain Structure
services-gatekeeper-domain
⊟-Environment
 ---Deployments
⊞-Services
 ---Security Realms
⊞-Interoperability
⊞-Diagnostics
 ---Media Server Control
 ---SipServer
⊟-OCSG
     ---AdminServer
     ---Alarms
     ---CDRs
     ---EDR Configuration
```

You monitor Services Gatekeeper configuration through the following nodes in the **Domain Structure** group for **services-gatekeeper-domain**, the container for all Services Gatekeeper servers:

- **Environment**, to create, configure, and control servers, clusters, server templates, virtual hosts, targets that can be migrated. Some configuration settings are cluster-wide, and other settings are per server.
- **Deployments**, to install, update, and delete installed applications. This link displays the list of Java EE applications and standalone application modules in this domain.

- **Services**, to configure WebLogic Server services.
- **Security Realms**, to configure the mechanisms that are used to protect WebLogic resources such as users, groups, security roles, security policies, and security providers. You can have multiple security realms in a WebLogic Server domain, but only one can be set as the default (active) realm.
- **Interoperability**, to configure interoperability between WebLogic Server applications and Tuxedo services.
- **Diagnostics**, to configure the collection of monitoring and performance data, through the WebLogic Diagnostics Framework (WLDF).
- **Media Server Control**, to provide media server connectivity. Applications can look up a media server control factory on the Java Naming and Directory Interface (JNDI) tree.
- **SipServer**, to access the Oracle Communications Converged Application Server Administration Console. To configure SIP settings, use the WebLogic Server Administration Console.

  See "Converged Application Server Administration Console for SIP-based Services" for information.
- **OCSG**:
  - *Server Name*- One entry per Services Gatekeeper server.
  - **Alarms** - displays the Alarms panel. See "Oracle Communications Services Gatekeeper Alarms Pane".
  - **CDRs** - displays the charging data records (CDRs) panel. See "Using the Services Gatekeeper CDRs Pane".
  - **EDR Configuration** - displays the EDR Configuration pane. See "Managing EDR/CDR/Alarm Configuration Files" for information.

Using the Configuration and Provisioning Pane

The **Configuration and Provisioning** pane contains two tabs:

- Attributes
- Operations

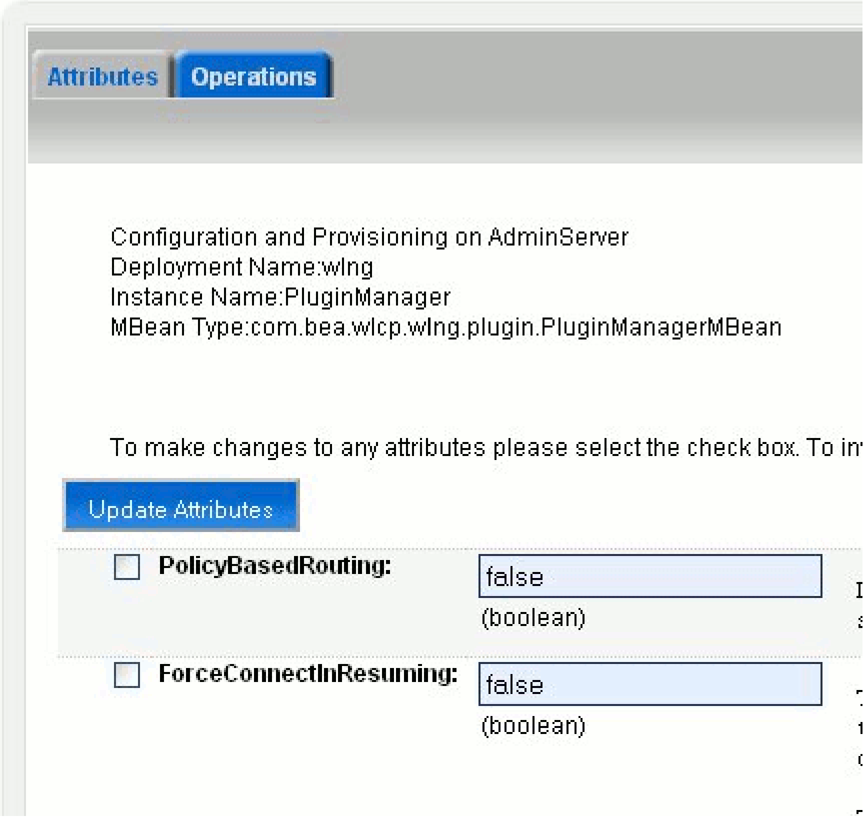
The **Attributes** tab displays a list of attributes, either read-only or read/write for the managed object. Attributes that have read/write capabilities have a check-box next to them.

> **Note:** This document identifies read-only attributes with an **(r)** after the name.

To change an attribute:

1. Select the check box.
2. Enter the new value in the entry field.
3. Click **Update Attributes**.

Figure 2–2 Example Configuration and Provisioning Pane Attributes Tab



Configuration and Provisioning on AdminServer
Deployment Name:wlng
Instance Name:PluginManager
MBean Type:com.bea.wlcp.wlng.plugin.PluginManagerMBean

To make changes to any attributes please select the check box. To in

□ **PolicyBasedRouting:** false (boolean)

□ **ForceConnectInResuming:** false (boolean)

The **Operations** tab contains a list with the methods that either display data, set data, or perform an actual task.

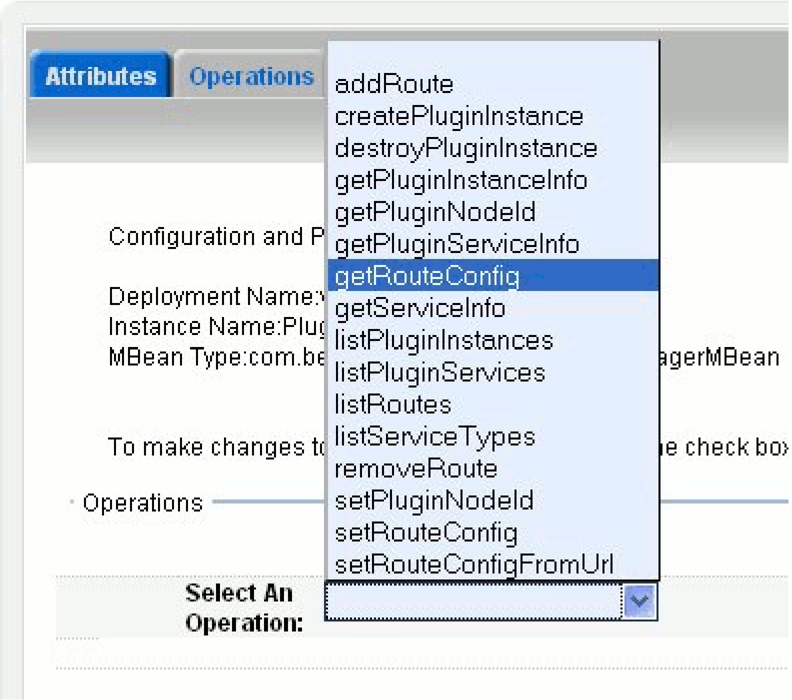
To call a method displayed in the **Operations** tab:

1. Select the method from the **Select An Operation** list.

   The fields display the information required by the method.

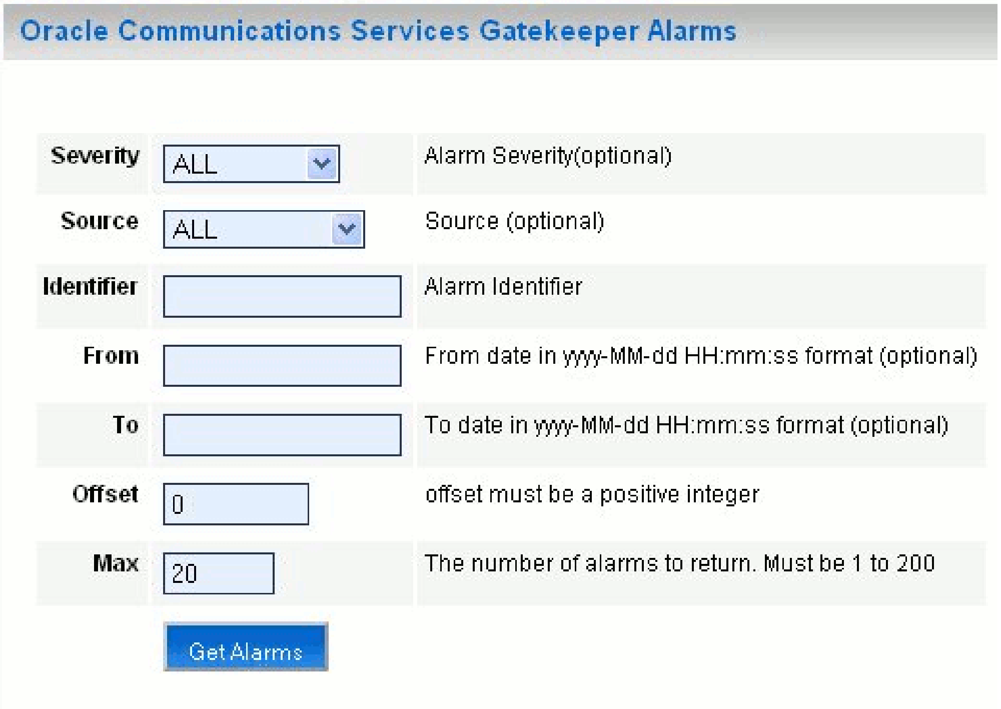
2. Enter the information in the fields.
3. To call the method, click **Invoke**.

Figure 2–3 Example Configuration and Provisioning Pane Operations Tab



Oracle Communications Services Gatekeeper Alarms Pane

Figure 2–4 Alarms Pane



Oracle Communications Services Gatekeeper Alarms

Severity	ALL	Alarm Severity(optional)
Source	ALL	Source (optional)
Identifier		Alarm Identifier
From		From date in yyyy-MM-dd HH:mm:ss format (optional)
To		To date in yyyy-MM-dd HH:mm:ss format (optional)
Offset	0	offset must be a positive integer
Max	20	The number of alarms to return. Must be 1 to 200

The Services Gatekeeper Alarms pane displays alarms emitted.

It is possible to filter the output to the page on a set of criteria. See Table 2–1.

To retrieve the current alarms, click **Get Alarms**.

Table 2–1 Services Gatekeeper Alarms Pane Display Filters

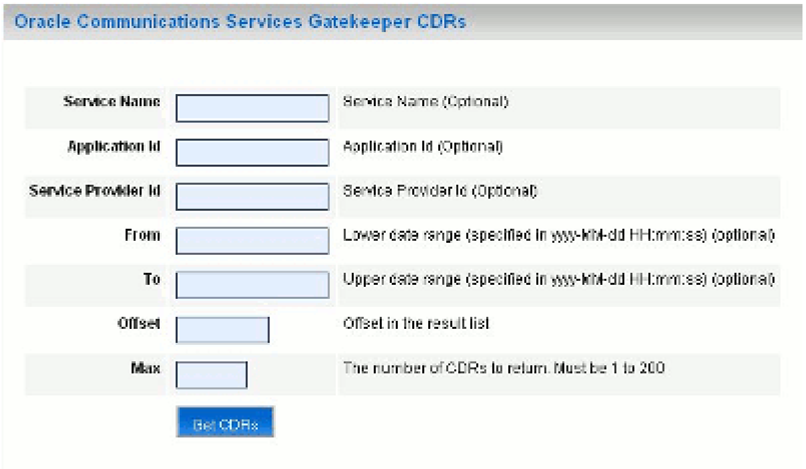
Filter on...	Input
Severity	From the **Severity** list, select which severity level to display. The options are: - **ALL** - **WARNING** - **MINOR** - **MAJOR** - **CRITICAL**
Source	From the **Server** list, select the server from which to display alarms. The options are: - **ALL**, for all servers - *Server_name*, for an individual server
Identifier.	See *Services Gatekeeper Alarms Handling Guide* for alarm identifiers. Use 0 (zero) to wildcard.
From	In the **From**, field, enter the start time for the time interval. The options are: - Exact time. - No given start time. Leave empty. Exact times are formatted as *YYYY-MM-DD hh:mm:ss*, where: - *YYYY* is the year. Four digits required. - *MM* is the month (1 through 12). - *DD* is the day (1 through 31). Upper limit depends on month and year. - *hh* is the hour (0 through 23). - *mm* is the minute (0 through 59). - ss is the second (0 through 59). **Note:** There must be a space separating *YYYY-MM-DD* and *hh:mm:ss*.
To	In the **To**, field, enter the end time for the time interval. The options are: - Exact time. - No given start time. Leave empty. Exact times are formatted as *YYYY-MM-DD hh:mm:ss*, where: - *YYYY* is the year. Four digits required. - *MM* is the month (1 through 12). - *DD* is the day (1 through 31). Upper limit depends on month and year. - *hh* is the hour (0 through 23). - *mm* is the minute (0 through 59). - ss is the second (0 through 59). **Note:** There must be a space separating *YYYY-MM-DD* and *hh:mm:ss*.

Table 2–1 (Cont.) Services Gatekeeper Alarms Pane Display Filters

Filter on...	Input
Offset	In the **Offset** field, enter the start offset in the list of alarms entries matching the criteria. Integer. 0 (zero) is the first entry.
Max	In the field **Max** field, enter the maximum number of alarm entries to return. Integer.

Using the Services Gatekeeper CDRs Pane

Figure 2–5 CDRs pane


Oracle Communications Services Gatekeeper CDRs
Service Name
Service Name (Optional)
Application Id
Application Id (Optional)
Service Provider Id
Service Provider Id (Optional)
From
Lower date range (specified in yyyy-MM-dd HH:mm:ss) (optional)
To
Upper date range (specified in yyyy-MM-dd HH:mm:ss) (optional)
Offset
Offset in the result list
Max
The number of CDRs to return. Must be 1 to 200
Get CDRs

The Services Gatekeeper Charging Data Records (CDRs) pane displays any generated CDRs.

It is possible to filter the output to the page on a set of criteria. See Table 2–2.

To retrieve the list of CDRs, click **Get CDRs**.

Table 2–2 Services Gatekeeper CDRs Pane Filters

Filter on...	Input
Service Name	In the **Service Name** field, enter the name of the service name for which CDRs are required. The service name is the service type defined for the network protocol plug-in. A blank entry indicates the wildcard character.
Application Id	In the **Application Id** field, enter the application ID to filter on. A blank entry indicates the wildcard character.
Service Provider Id	In the **Service Provider Id** field, enter the service provider ID to filter on. A blank entry indicates the wildcard character.

Table 2–2 (Cont.) Services Gatekeeper CDRs Pane Filters

Filter on...	Input
From	In the **From** field, enter the start time for the time interval. The options are: ■ Exact time. ■ No given start time. Leave empty. Exact times are formatted as *YYYY-MM-DD hh:mm*:ss, where: ■ *YYYY* is the year. Four digits required. ■ *MM* is the month (1 through 12). ■ *DD* is the day (1 through 31). Upper limit depends on month and year. ■ *hh* is the hour (0 through 23). ■ *mm* is the minute (0 through 59). ■ ss is the second (0 through 59). **Note:** There must be a space separating *YYYY-MM-DD* and *hh:mm:ss*.
To	In the **To** field, enter the end time for the time interval. The options are: ■ Exact time. ■ No given start time. Leave empty. Exact times are formatted as *YYYY-MM-DD hh:mm*:ss, where: ■ *YYYY* is the year. Four digits required. ■ *MM* is the month (1 through 12). ■ *DD* is the day (1 through 31). Upper limit depends on month and year. ■ *hh* is the hour (0 through 23). ■ *mm* is the minute (0 through 59). ■ ss is the second (0 through 59). **Note:** There must be a space separating *YYYY-MM-DD* and *hh:mm:ss*.
Offset	Enter the start offset in the list of CDR entries matching the criteria. Enter integer values. 0 (zero) is the first entry.
Max	Enter the maximum number of CDR entries to return. Enter integer values.

Converged Application Server Administration Console for SIP-based Services

You use the Oracle Communications Converged Application Server Administration Console to configure settings for that product and the SIP Servlet parts of Services Gatekeeper communication services.

To open the Converged Application Server Administration Console, click *Domain Name*, and then **SipServer**.

Localizing the Administration Console

For instructions on setting the Administration Console to a different locale, see “Rebranding the Administration Console” in *Oracle Fusion Middleware Extending Administration Console for Oracle WebLogic Server*.

Understanding the Java Management MBeans

Services Gatekeeper exposes its management interfaces as Java Management Extensions (JMX) MBeans.

These MBeans come in two types:

- Standard MBeans
- Configuration MBeans

> **Note:** A link to Javadoc for Oracle Access Manager (OAM) can be found in the Reference topic page in the Services Gatekeeper documentation set. See the reference section for each OAM service for the fully qualified MBean names and information about how they map to managed objects in the Services Gatekeeper Administration Console.

The code in Example 2–1 connects to the MBean server and calls a method to complete a task on the JMX interfaces.

The MBean object name is associated with a version number. The MBean name includes the version number for the release. Update external JMX clients according to the release number.

Example 2–1 Example of using JMX to manage Services Gatekeeper

```
import java.io.IOException;
import java.net.MalformedURLException;
import java.util.Hashtable;
import javax.management.MBeanServerConnection;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;
import javax.naming.Context;
public class TestMgmt {
  private static MBeanServerConnection connection;
  private static JMXConnector connector;
/*
  * Initialize connection to the Domain Runtime MBean Server
  */
  public static void initConnection(String hostname, String portString,
                                   String username, String password) throws IOException,
      MalformedURLException {
    String protocol = "t3";
    Integer portInteger = Integer.valueOf(portString);
    int port = portInteger.intValue();
    String jndiroot = "/jndi/";
    String mserver = "weblogic.management.mbeanservers.domainruntime";
    JMXServiceURL serviceURL = new JMXServiceURL(protocol, hostname,
        port, jndiroot + mserver);
    Hashtable h = new Hashtable();
    h.put(Context.SECURITY_PRINCIPAL, username);
    h.put(Context.SECURITY_CREDENTIALS, password);
    h.put(JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES,
        "weblogic.management.remote");
    connector = JMXConnectorFactory.connect(serviceURL, h);
    connection = connector.getMBeanServerConnection();
  }
  public static void main(String[] args) throws Exception {
    String hostname = args[0];   //hostname of the admin server
    String portString = args[1]; //port of the admin server
    String username = args[2];
    String password = args[3];
    String mbserverName = args[4];  //NT server name
    String operationName = "addRoute";
    String id = "Plugin_px21_multimedia_messaging_mm7";
    String addressExpression = ".*";
    String[] params = new String[]{id, addressExpression};
    String[] signature = new String[]{"java.lang.String", "java.lang.String"};
    ObjectName on;
    try {
      on = new ObjectName("com.bea.wlcp.wlng:Name=wlng,InstanceName=PluginManager,Type=
com.bea.wlcp.wlng.plugin.PluginManagerMBean,Location=" + mbserverName);
    } catch (MalformedObjectNameException e) {
      throw new AssertionError(e.getMessage());
    }
    initConnection(hostname, portString, username, password);
    //invoke the operation
    Object result = connection.invoke(on, operationName, params, signature);
    System.out.println(result.toString());  //displays the result
    connector.close();
  }
}
```

If an attempt to set an attribute or call a method fails, an **com.bea.wlcp.wlng.api.management.ManagementException** exception, or a subclass of this exception is thrown. The following subclasses are defined:

- **DuplicateKeyException**, thrown when the called method creates a duplicate key.
- **InputManagementException**, thrown for invalid user input.
- **KeyNotFoundException**, thrown when the called method cannot find the specified key.

If a JMX client does not have the same version of exception classes available, the client throws **ClassNotFoundException**.

The *Middleware_Home*/**ocsg_pds/lib/wlng/oam.jar** file contains custom classes for OAM and return types. The *Middleware_Home*/**ocsg_pds/doc/javadoc_oam** file contains Javadoc for OAM. The *Services Gatekeeper OAM Java API Reference* is also available at the top level of this documentation set.

Using the WebLogic Scripting Tool to Administer Services Gatekeeper

The following section gives examples of how to use the WebLogic Scripting Tool (WLST) in both interactive and script mode when configuring and managing Services Gatekeeper.

For information about WebLogic Server and WLST, see “Introduction and Roadmap” in *Oracle WebLogic Server Understanding the WebLogic Scripting Tool*.

The following topics are covered in this section:

- Using WLST in Interactive Mode
  - Connecting to Services Gatekeeper with WLST

> **Note:** A link to Javadoc for OAM can be found on the Reference topic page in the Services Gatekeeper documentation set.

Using WLST in Interactive Mode

This section gives examples of how to use WLST in interactive mode.

Connecting to Services Gatekeeper with WLST

To connect to Services Gatekeeper with WLST:

1. To ensure that the Java environment is set, enter the following command:

   – On UNIX systems:

     Domain_Home/**bin/setDomainEnv.sh**

   – On Windows systems:

     Domain_Home\bin\\setDomainEnv.cmd

2. To start WLST, enter the following command:

   ```
   java weblogic.WLST
   ```

3. To connect to the server to manage, enter the following command:

   `connect('`*username*`','`*password*`','t3://`*host*`:`*port*`')`

4. To change to the custom tree where Services Gatekeeper MBeans are located, enter:

   ```
   custom()
   ```

   ```
   cd('com.bea.wlcp.wlng')
   ```

5. To display a list of the Mbeans, enter:

   ```
   ls()
   ```

   The MBean names are also displayed in each **Configuration and Provisioning** page for the management objects in the Services Gatekeeper Administration Console.

6. To select the MBean:

   `cd('com.bea.wlcp.wlng:Name=wlng_nt,Type=`*MBean_name*`')`

   For example, to select the MBean for the EDRService, use:

   ```
   cd('com.bea.wlcp.wlng:Name=wlng,InstanceName=EdrService,Type=com.bea.wl
   cp.wlng.edr.management.EdrServiceMBean')
   ```

Disconnecting from WLST

To disconnect from Services Gatekeeper:

1. Enter:

```
disconnect()
```

2. To exit from the WLST shell, enter the following command:

   ```
   exit()
   ```

Changing an MBean Attribute

To change the attribute of an MBean:

1. Select the MBean.
2. Set the attribute:

   set('*name_of_attribute*',*value_of_attribute*)

   For example, to change the attribute `BatchSize` to 2001 in the managed object EDRService, enter:

   ```
   set('Batchsize', 2001)
   ```

To display the attribute values for an MBean, enter:

```
ls()
```

Calling an MBean Method

To call an MBean method:

1. Select the MBean.
2. Define the parameters as an array.
3. Define the data types as an array.
4. Call the method.

The `displayStatistics` method in the EDRService MBean takes no arguments. For example:

```
cd('com.bea.wlcp.wlng:Name=wlng,InstanceName=EdrService,Type=com.bea.wlcp.wlng.edr
.management.EdrServiceMBean')
objs = jarray.array([],java.lang.Object)
strs = jarray.array([],java.lang.String)
print(invoke('displayStatistics',objs,strs))
```

To add a route using the MBean for the Plug-in Manager, call the **addRoute** method. The **addRoute** method takes two arguments. State the values of the parameters in the order in which the parameters are defined in the signature of the method. For example:

```
cd('com.bea.wlcp.wlng:Name=wlng,InstanceName=PluginManager,Type=com.bea.wlcp.wlng.
plugin.PluginManagerMBean ')
objs=jarray.array(['Plugin_px21_multimedia_messaging_mm7','.*'],Object)
strs=jarray.array(['java.lang.String', 'java.lang.String'],String)
invoke('addRoute', objs, strs)
```

For a method that has input parameters and a string return value, run the **invoke** command as shown below:

```
objs =
jarray.array([java.lang.String('stringInput1'),java.lang.String('stringInput2'),
java.lang.String('StringInput3')],java.lang.Object)
strs = jarray.array(['java.lang.String', 'java.lang.String',
'java.lang.String'],java.lang.String)
```

```
invoke('methodName',objs,strs)
```

Scripting WLST

To run WLST using scripts, use the following command:

```
java weblogic.WLST script_name.py argument_1 argument_2 ...argument_n
```

The arguments can be retrieved from within the scripts in the array **sys.argv[]**, where **sys.argv[0]** is the script name, **sys.argv[1]** is the second argument and so on. You can also specify the login information and connection information as arguments to the python script.

The script in Example 2–2 connects to Services Gatekeeper and changes an MBean defined by argument.

Example 2–2 Example of script

```
userName = sys.argv[1]
passWord = sys.argv[2]
url="t3://"+sys.argv[3]+":"+sys.argv[4]
objectName = sys.argv[5]
objectName = "com.bea.wlcp.wlng:Name=nt,Type=" + sys.argv[5]
print objectName
connect(userName, passWord, url)
custom()
cd('com.bea.wlcp.wlng')
cd(objectName)
```

The following command uses the script:

```
java weblogic.WLST script1.py weblogic weblogic localhost 7001
com.bea.wlcp.wlng:Name=wlng,InstanceName=PluginManager,Type=com.bea.wlcp.w
lng.plugin.PluginManagerMBean
```

Understanding Services Gatekeeper Database Maintenance

This section lists the database tables that you clean periodically and provides an example script.

About Cleaning Database Tables

Table 2–3 lists the database tables that you must periodically clean to prevent them from growing too large and adversely affecting performance. See your database documentation for instructions on deleting rows from these tables.

Table 2–3 Database Table Cleaning Intervals

Type of Database Table	Table	Recommended Cleaning Interval
Services Gatekeeper	SLEE_ALARM	Every two months
Services Gatekeeper	SLEE_CHARGING	See "About Cleaning the SLEE_CHARGING Table" for details.
Services Gatekeeper	SLEE_STATISTICS_DATA	Every month
EDR Analytics	TRANSACTION_HISTORY	Daily if used. See "About Cleaning EDR Analytics Tables" for details.

Table 2–3 (Cont.) Database Table Cleaning Intervals

Type of Database Table	Table	Recommended Cleaning Interval
EDR Analytics	TRANSACTION_PARAMETER_HISTORY	Daily if used. See "About Cleaning EDR Analytics Tables" for details.
EDR Analytics	TRANSACTION_INVALID_HISTORY	Daily if used. See "About Cleaning EDR Analytics Tables" for details.
EDR Analytics	TRANS_PARA_INVALID_HISTORY	Daily if used. See "About Cleaning EDR Analytics Tables" for details.

About Cleaning EDR Analytics Tables

Clean the EDR analytics tables listed in Table 2–3 if you use the EDR analytics feature and have **EdrAanalyticMBean.StoreHistory** set true. The cleaning interval depends on the amount of traffic stored in your database. However, these tables tend to grow quickly and require daily maintenance.

About Cleaning the SLEE_CHARGING Table

The SLEE_CHARGING table cleaning interval depends on the amount of charging traffic, so it is difficult to specify a recommended interval. There is usually one row in this table per plug-in transaction. A row can contain up to 300 bytes of data.

Operators export charging data from the SLEE_CHARGING table to a file either by using a scheduled script or by integrating with a billing system such as Oracle Communications Billing and Revenue Management.

Charging data is contained in CDRs, which are a type of event data record (EDR). CDRs are integrated by EDR listeners. You can also listen for alarms by setting up an SNMP/EDR listener. If you do so, disable CDR and alarm storage in the database by setting the **StoreAlarms** or **Store CDRs** attributes to `false` for the EDR Service.

> **Note:** Oracle recommends storing alarms for some period, for trouble-shooting purposes.
>
> See the **StoreCDRs** and **StoreAlarms** fields of the **EdrServiceMBean** in the "All Classes" section of *Services Gatekeeper OAM Java API Reference* for details.

For more information about

- EDRs, CDRs, and alarms, see "Managing and Configuring EDRs, CDRs and Alarms".
- "Creating EDR Listeners", see *Services Gatekeeper Extension Developer's Guide*.
- Generating SNMP MIB Files, see "Generating SNMP MIB Files".
- Setting the expiration of data stored in the database tables, see "Configuring Storage Expiration".

Creating a Script to Clean SLEE_CHARGING

If your Services Gatekeeper implementation has high traffic, such that these three tables are updated with millions of records per day, Oracle recommends that you create a script to clean them and run it for two to three hours. This script would ideally copy all data that arrived after the last cleaning to another table and then delete everything currently in the table. This script does not require a server restart.

To clean the SLEE_CHARGING table, create a shell script or PERL/SQL combination such as the following:

1. Set up a repository for the data in the SLEE_CHARGING data. To do so, create a table called SLEE_CHARGING_BACKUP.
2. Create a SQL query that:
   1. Selects all data in the SLEE_CHARGING table that is more than 15 minutes old.

      For example, if you run the script every 3 hours, the script would capture 2 hours and 45 minutes worth of data.
   2. Insert the selected data/rows into the SLEE_CHARGING_BACKUP table.
   3. Delete the same data from the SLEE_CHARGING table.

The steps a and b can easily be combined in one SQL query, for example:

```
insert into slee_charging_bkup(list all columns~ )
select list all the columns~
from slee_charging
where stored_ts millisecs calculated~
```

Archive the data into a new table called SLEE_STATISTICS_DATA as shown in Example 2–3:

Example 2–3 Archiving Data from the SLEE_CHARGING Table

```
create table slee_statistics_data_tmp1(
slee_name,
timestamp,
type_id,
transactions,
source_entity,
source_app_id,
datetime,
primary key ( timestamp, type_id )
)
organization index
nocompress
as select slee_name, timestamp, type_id, transactions, source_entity, source_app_
id,
to_date(to_char(timestamp'1970-01-01 00:00:00.00' +
numtodsinterval(timestamp/1000,'SECOND'),
'DD-MON-YYYY HH24:MI:SS'),'DD-MON-YYYY HH24:MI:SS') from slee_statistics_data;
```

3

Starting, Stopping, and Administering Servers

This chapter describes how to start and stop servers in an Oracle Communications Services Gatekeeper domain, and configuring the heartbeats between servers.

Overview of Starting and Stopping Servers

A typical production Services Gatekeeper domain contains multiple access tier and network tier servers, with dependencies among the different types of servers. The process of starting up a domain is given here:

1. Start the administration server for the domain.

   The administration server provides the initial configuration to access tier and network tier servers in the domain. It can also be used to monitor the startup/shutdown status of each managed server.

   Start the administration server by using either the **startAdminServer** script installed with the Configuration wizard or a custom startup script.

2. Start network tier servers in each partition.

   The access tier cannot function until servers in the network tier are available.

3. Start access tier (application tier) servers in each partition.

> **Caution:** All servers should be started and available before opening the system to production network traffic.

Performance Implications

The Services Gatekeeper start scripts use default values for many Java Virtual machine (JVM) parameters that affect performance. For example, JVM garbage collection and heap size parameters may be omitted or may use values that are acceptable only for evaluation or development purposes. In a production system, you must rigorously profile your applications with different heap size and garbage collection settings in order to realize adequate performance.

Starting Service Gatekeeper Servers

To start Services Gatekeeper servers, you can use the following:

- Scripts. See "Starting and Stopping Services Gatekeeper Servers with Scripts".
- Node Manager Utility. See "Starting Servers with the WebLogic Node Manager Utility".

- Other methods.

  For other ways of starting servers, see "Starting and Stopping Servers" in *Oracle WebLogic Server Managing Server Startup and Shutdown for Oracle WebLogic Server*.

Special Instruction for Starting Servers on Solaris 64-bit Systems

If your installation supports Solaris 64-bit and you are using Sun’s JVM, you must add the `-d64` flag to whichever startup script you are using. If you do not use this flag, the JVM will default to 32-bit.

Starting and Stopping Services Gatekeeper Servers with Scripts

Start network tier and access tier servers by using either the **startManagedWebLogic** script installed with the Configuration wizard or your own custom startup script.

To use the **startManagedWebLogic** script, specify the name of the server to start and the URL of the administration server for the domain using the syntax:

```
startManagedWebLogic.sh managed_server_name admin_url
```

For example:

```
startManagedWebLogic.sh networknode0-0 t3://adminhost:7001
```

About User Credentials for WebLogic Servers

By default, the servers are started in production mode. This means that user credentials must be provided. For more information, see the discussion about:

- How to "Provide User Credentials to Start and Stop Servers" in *Oracle Fusion Middleware Managing Server Startup and Shutdown for Oracle WebLogic Server* (Release 12c).
- “Development vs Production Mode Default Tuning Values" in *Oracle Fusion Middleware Performance and Tuning for Oracle WebLogic Server* (Release 12c).
- "Node Manager Overview" in *Oracle Fusion Middleware Node Manager Administrator’s Guide for Oracle WebLogic Server* (Release 12c) for details.

Starting Servers with the WebLogic Node Manager Utility

You can also start network tier and access tier servers by using the Administration Console with an instance of Node Manager running on each machine. There are many different ways to run Node Manager.

A simple Java-based version follows:

> **Note:** Windows is not supported for production servers.
>
> The following instructions assume a UNIX or Linux system. The instructions must be followed on a managed server.

1. To properly start Services Gatekeeper, configure the **nodemanager.properties** file on each node.

   To start managed servers in clustered environments using Node Manager, edit **nodemanager.properties** to use the startup script during Services Gatekeeper domain configuration. Using this startup script ensures that the **classpath** is set correctly so managed servers have access to Services Gatekeeper files.

To configure **nodemanger.properties**:

   a. Open **nodemanager.properties** in a text editor.

   b. Ensure `StartScriptEnabled` is set to **true**.

   c. Ensure `StartScriptName` specifies the Services Gatekeeper startup script on the node.

2. Start the node manager.

   The best practice is to have this as part of the normal machine startup sequence. To do it manually:

   a. Log in into the server.

   b. Change to the *Middleware_home*/**wlserver/server/bin** directory.

   c. Run the `./startNodeManager.sh` script.

3. Add the domains that the Node Manager instance controls to the *Middleware_home*/**wlserver_10.3/common/nodemanager/nodemanager.domains** file.

   See "Configuring nodemanager.domains File" in *Oracle Fusion Middleware Node Manager Administrator's Guide for Oracle WebLogic Server* for a description of **nodemanager.domains**.

   A sample entry for a domain in this file is:

   ```
   gatekeeper-domain=/bea/user_projects/domains/gatekeeper-domain
   ```

4. Edit the *Middleware_home*/**wlserver/common/nodemanager/nodemanager.properties** file. Ensure that `StartScriptEnabled=true` is set.

5. Restart the node manager, using Step 1 above.

6. To create a startup script, do the following:

   a. In the *domain_home* directory, create a file called **startWeblogic.sh**.

   b. Add the following line:

      ```
      ./bin/startManagedWebLogic.sh SERVER_NAME ADMIN_HOST_PORT
      ```

      For example: `./bin/startManagedWebLogic.sh NT1 192.168.1.42:7001`

7. Ensure that **Listen Address** is configured in the Administration Console. To do so:

   In **Domain Structure**, click **Environment,** then **Machines**, then *machine_name*, and finally **Node Manager.**

   You must click **Lock & Edit** before you make any changes. Click **Save**, if you make any changes.

8. To start Services Gatekeeper servers using the Node Manager:

   a. Go to the domain's Administration Console.

   b. Under **Environment**, select the managed servers you want to start.

Configuring Domain Server Heartbeats

Services Gatekeeper uses a heartbeat mechanism to continuously ensure that all NT/AT servers in a cluster are available to process traffic. This feature is not enabled by default; you enable it from the Administration Console using the **IsActive** field of the **HeartbeatMBean**.

See "HeartBeatMBean Reference" for information on how to use **HeartBeatMBean**..

The heartbeat mechanism is designed to ensure the high availability (HA) access among servers in a cluster. This feature sends heartbeat messages to all servers in a cluster, and if one does not respond within a configurable time, it reroutes all traffic to servers that are available. This mechanism replaces the Oracle WebLogic Server heartbeat mechanism which uses the NT server JNDI tree for routing.

The heartbeat messages are very small and the default settings have a negligible affect on server performance if enabled.

Server communication may fail for many reasons, including being shutdown for maintenance, crashing, or for network problems unrelated to the server itself. The heartbeat mechanism just determines whether the server is available to process traffic. If all servers become unavailable and no traffic can be processed, the WebLogic server itself sends an error message.

For an overview of the Services Gatekeeper high availability features, see “About High Availability” in *Services Gatekeeper Concepts*.

Services Gatekeeper also includes an unrelated network node heartbeat mechanism that plug-ins use to determine if network nodes are functional. See "Configuring Network Node Heartbeats" for details.

Understanding the Services Gatekeeper Heartbeat Settings

Once enabled, you use the heartbeat settings in the Administration Console (**oracle.ocsg.heartbeat.nt.management** object; **HeartbeatMBean** MBean) to control how Services Gatekeeper decides a server is unavailable for processing traffic. A server is considered unavailable if it has not responded to one of the 3 consecutive heartbeat messages within 900 milliseconds (300 milliseconds per message). If a server is unavailable, no further traffic is sent to it. The probing server continues to send heartbeat requests to the unavailable server, and returns it to service if it meets the response criteria. The number of heartbeats and the time to respond to each are configurable.

The **HeartbeatMBean** MBean does not generate alarms, EDRs, or error messages. See "HeartBeatMBean Reference" for information on how to use **HeartBeatMBean**.

HeartBeatMBean Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services**, followed by **HeartBeat**. You can also set them using a Java application. For information on the methods and fields, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Stopping Service Gatekeeper Servers

It is recommended that you shut down WebLogic Server instances through the Administration Console. See the instructions to shut down a server instance, control graceful shutdowns, and shut down servers in a cluster in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Help*.

4

Managing, Backing Up, and Restoring Services Gatekeeper

This chapter describes managing Oracle Communications Services Gatekeeper to prevent server failure, and backing up and restoring critical application files and data.

About Managing, Backing Up and Restoring Services Gatekeeper

Services Gatekeeper uses Oracle WebLogic Server as its application server and Oracle Enterprise Database or Oracle MySQL Database as its storage repository. Managing, backing up and restoring Services Gatekeeper in your environment follows the general guidelines and best practices used when ensuring Oracle WebLogic Server, Oracle Enterprise Database, and Oracle MySQL Database can recover from server failure or other disaster.

See "Avoiding and Recovering From Server Failure" in *Fusion Middleware Manager Startup and Shutdown for Oracle WebLogic Server* for more information on ensuring that the WebLogic Server component of your Services Gatekeeper installation is properly configured to avoid failures, and recover from them in the event one occurs.

If you store your Services Gatekeeper data in an Oracle Enterprise Database, see the section on backup and recovery in the *Oracle Database Documentation Library* for more information about backing up and restoring Services Gatekeeper data.

- For Oracle Enterprise Database 11*g* R2 users, see "Backup and Recovery User's Guide" here:

  `http://docs.oracle.com/cd/E11882_01/backup.112/e10642/toc.htm`

  and "*Backup and Recovery Reference*" here:

  `http://docs.oracle.com/cd/E11882_01/backup.112/e10643/toc.htm`

  for more information.

- For Oracle Enterprise Database 12c users, see "*Backup and Recovery User's Guide*" here:

  `http://docs.oracle.com/cd/E16655_01/nav/portal_4.htm#backup_and_recovery`

  and "*Backup and Recovery Reference*" here:

  `http://docs.oracle.com/cd/E16655_01/backup.121/e17631/toc.htm`

  for more information.

See "*MySQL Backup and Recovery*" here:

http://dev.mysql.com/doc/refman/5.7/en/backup-and-recovery.html

for more information about backing up and restoring Services Gatekeeper data if you store your data in an Oracle MySQL database.

5

Securing Services Gatekeeper

This chapter explains the Oracle Communications Services Gatekeeper system administration security issues.

About Services Gatekeeper Security

This chapter explains security issues that affect Services Gatekeeper system administrators, including these sections:

- About Services Gatekeeper Communication Security
- About Protecting Your Network
- About Filtering Tunneled Parameters
- Securing SOAP-Based Web Services with WS-Security
- Using SAML Tokens Between RESTful Applications
- Securing RESTful Web Services with SSL
- Securing Network-Facing Servers With Keystores
- Securing the Oracle Access Manager MBeans

Also see *Services Gatekeeper Security Guide* for additional information on security, including these sections:

- Services Gatekeeper Security Overview
- Performing a Secure Services Gatekeeper Installation
- Deploying Services Gatekeeper in a Demilitarized Zone
- Implementing Services Gatekeeper Security
- Deploying Services Gatekeeper in a Demilitarized Zone
- Securing Communication Services
- Securing Partner Relationship Management Portals

About Services Gatekeeper Communication Security

One of the first things you must do while setting up your Services Gatekeeper implementation is decide how to protect your web service communication and OAM MBeans.

For more information about Services Gatekeeper security and how these tasks help create a secure Services Gatekeeper implementation, see *Services Gatekeeper Security Guide*.

Understanding Web Services Security

Web service security controls the network traffic between Services Gatekeeper and applications and network nodes it communicates with. By default, Services Gatekeeper is configured to send credentials in clear text for inbound traffic only. Outbound security is not enabled because there is no way for Services Gatekeeper to know what security policies may be required by a client. Set up that level of security at the earliest after you find out what your clients require.

Oracle recommends that for production implementations you fully secure web services using the instruction in one of these sections, depending on the web interface you use:

- Securing SOAP-Based Web Services with WS-Security
- Securing RESTful Web Services with SSL
- Securing Network-Facing Servers With Keystores

Secure client (outbound) traffic and require digest authentication (encrypted credentials) for all traffic. Applying a security policy (WS-Security) to a web service establishes both inbound and outbound security policies. To secure the link by which Services Gatekeeper returns notifications, use Secure Sockets Layer (SSL).

Understanding OAM MBean Security

MBean security controls who is authorized to modify the runtime MBean Server within your WebLogic Server installation. This server allows OAM procedures to be performed.

Oracle recommends that you secure the OAM MBeans using the instructions in "Securing the Oracle Access Manager MBeans".

About Protecting Your Network

Services Gatekeeper provides a set of firewall settings to protect against denial of service (DOS) attacks from malicious or malformed REST or SOAP messages. For details see "*Security Considerations for Developers*" in *Services Gatekeeper Security Guide*.

About Filtering Tunneled Parameters

Some communication services use tunneled parameters to send information to Services Gatekeeper, which can present a security vulnerability. You can use an interceptor to filter tunneled parameters to control this vulnerability and limit which xparameters can be set by inbound messages.

Filtering blocks requests that contain xparameters that are not specifically configured as allowed. This filtering applies to all of Services Gatekeeper, not just individual applications.

For general information about xparameters and filtering, see the discussion on using parameter tunneling in the *Services Gatekeeper Extension Developer's Guide*.

Configuring Tunneled Parameters Filtering

This section explains the behavior and configuration of the tunneled parameters filtering application.

About the XParameter Filter Application

When enabled, the **interceptor_xparam** application filters inbound messages for tunneled xparameters in inbound messages. Messages containing xparameters not explicitly allowed in the application's configuration file are rejected. Specifying a list of allowable xparameters enables tighter security in your Services Gatekeeper implementation. Filtering is on a global, not application, level.

The interceptor_xparam application is installed with Services Gatekeeper and is deployed as a standalone EAR file named **xparam_interceptors.ear**. It is implemented by the **x-param-filter-interceptor** interceptor.

To turn the application on, configure the customized interceptor chain and enable **interceptor_xparam**. You must configure the xparameters that you want to allow as described in "XParameter Filter Configuration File"; otherwise all requests that contain xparameters are rejected. If the application is active when Services Gatekeeper starts up, the application reads a configuration file that lists the allowed xparameters. If the list of allowed xparameters changes, you must update the configuration file and redeploy the filtering application. You can disable all tunneled parameters filtering by stopping or undeploying the EAR file and removing the interceptor from your interceptor chain.

XParameter Filter Configuration File

The xparameter filter configuration file, **xparam_filter_config.xml**, is located in the **xparam_interceptors.ear** in the **APP-INF/classes** directory. The schema file, **xparam_filter_config.xsd**, is also available in this directory.

To allow an xparameter in a request, list its key as an `<xParamKey>` sub element within the `<allowedXParams>` element in **xparam_filter_config.xml**. For example:

```
<allowedXParams>
 <xParamKey>sms.protocol.id</xParamKey>
 <xParamK>sms.service.type<xParamKey>
 . . .
</allowedXParams>
```

The xparameter keys are listed by communication service in the sections on tunneled parameters in the chapters in the *Services Gatekeeper Communication Service Reference Guide*. Some communication services do not support any xparameters.

XParameter Rejection

If an xparameter is not configured in **xparam_filter_config.xml**, a SOAP request that passes the xparameter is rejected.

The following is an example of a rejection response to a request that passed the xparameter called `dest_addr_subunit`:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Header/>
 <env:Body>
    <env:Fault>
       <faultcode>env:Server</faultcode>
       <faultstring/>
       <detail>
```

```
          <v2:ServiceException
xmlns:v2="http://www.csapi.org/schema/parlayx/common/v2_1">
             <messageId>SVC0001</messageId>
             <text>A service error occurred. Error code is %1</text>
             <variables>Error validating the request, xparam: dest_addr_subunit in
the request data is not allowed.</variables>
          </v2:ServiceException>
       </detail>
    </env:Fault>
 </env:Body>
</env:Envelope>
```

See "Internal XParameters" for information about exceptions for xparameters that are not rejected.

Internal XParameters

It is possible for custom interceptors to insert xparameters on behalf of an application. To prevent these internal xparameters from being rejected by the filter, ensure that any custom interceptor that adds xparameters is executed after the **x-param-filter-interceptor** in the interceptor chain.

Xparameters added to a request through the `<contextAttribute>` of an SLA are not rejected because the **InjectValuesInRequestContextFromSLA** interceptor executes after **x-param-filter-interceptor**.

See "Using Service Interceptors to Manipulate Requests" in the *Services Gatekeeper Extension Developer's Guide* for information about the order of execution of custom interceptors.

Securing SOAP-Based Web Services with WS-Security

SOAP-based security provides end-to-end message-level security for web services through an implementation of the WS-Security standard.

WS-Security defines a mechanism that offers three levels of security to SOAP messages:

- Authentication tokens (username token). The digest authentication version of authentication tokens is the default SOAP-based communication security setting. WS-Security authentication tokens let an application provide a user name and password or X.509 certificate for the purpose of authentication headers. With additional setup, Security Assertion Markup Language (SAML) can also be used for authentication.

  SOAP-based communication with the Partner Relationship Manager uses this security by default. However you must configure other web services to use this strategy. See "Setting up UsernameToken with Password Digest" for instructions.

- XML encryption (SAML tokens). WS-Security's use of W3C's XML encryption standard enables the XML body or portion of it to be encrypted to ensure message confidentiality.

  To set up SAML token security, configure the WebLogic SAML Identity Assertion Provider, which authenticates users based on SAML assertions and SAML credential mapping provider. The SAML Identity Assertion Provider is required only if you are using SAML assertions. See "Using Security Assertion Markup Language (SAML) Tokens For Identity" in *Oracle WebLogic Server Securing WebLogic Web Services for Oracle WebLogic Server.*

- XML digital signatures (X.509 certificate tokens). WS-Security's use of W3C's XML digital signatures lets the message be digitally signed to ensure message integrity. The signature is based on the content of the message itself (by applying the hash function and public key). If the message is altered en route, the signature becomes invalid. See "Setting up UsernameToken with X.509" for instructions.

Services Gatekeeper uses a WebLogic Server mechanism for web services Security - WSSE (Web Services Security Environment) policies:

- "Oracle Web Services Security" in *Oracle WebLogic Server Developing Applications with Oracle Security Developer Tools*.
- "Oracle WSM Policy Framework" in *Oracle WebLogic Server Developing Applications with Oracle ADF Data Controls.*
- "References" (specification references) in *Oracle WebLogic Server Developing Applications with Oracle Security Developer Tools.*

Authentication is handled transparently by WS-Security and subsequently by the configured authentication providers and login modules of the WebLogic Security framework. WS-Security also supports signing and encrypting a message by providing a security token hierarchy associated with the keys used for signing and encryption (for message integrity and confidentiality).

Setting up UsernameToken with Password Digest

To apply a WSSE policy of the type UsernameToken with Password Digest to a web service endpoint in Services Gatekeeper from the Administration Console:

1. Start the Administration Console.
2. Enable digest authentication:
   a. Click **Providers**.
   b. Click **Authentication**.
   c. Select the **WLNG Application Identity Asserter** check box.
   d. Click the **WLNG Application Identity Asserter** link.

      The **Settings for WLNG Application Identity Asserter** window appears.
   e. Under **Active Types:** move **wssePasswordDigest** from **Available:** to **Chosen:** using the left arrow icon.
   f. Click **Save**.
3. Create a custom `ws-policy.xml` file for the password digest. See Example 5–1 for an example.
4. For every web service:
   a. Put a copy of the custom `ws-policy.xml` file in the `WEB-INF/policies` directory of the WAR file for the web service.
   b. Edit the `WEB-INF/policies/weblogic-webservices-policy.xml` file replacing the old wsl-policy file with `policy:UsernameTokenDigestPolicy.xml`.
   c. Repackage and redeploy these files.
   d. Edit the deployment plan, `plan.xml`, to indicate inbound only for entry `WsPolicy_policy: UsernameTokenDigestPolicy.xml` in `plan.xml`.

Example 5–1 Username Token with digest custom policy example (UsernameTokenDigestPolicy.xml)

```
<?xml version="1.0"?>
<!-- WS-SecurityPolicy -->
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
  xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
  >
  <!-- Identity Assertion -->
  <wssp:Identity>
    <wssp:SupportedTokens>
      <!-- Use UsernameToken for authentication -->
      <wssp:SecurityToken IncludeInMessage="true"

TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#Usern
ameToken">
        <wssp:UsePassword

Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDi
gest"/>
      </wssp:SecurityToken>
    </wssp:SupportedTokens>
  </wssp:Identity>
</wsp:Policy>
```

Setting up UsernameToken with X.509

This section outlines how to set up WSSE with X.509, configure the default identity asserter, and configure the keystore.

To set up UsernameToken with X.509 from the Administration Console:

1. Select **Security Realms** from the **Domain Structure** menu for the domain you want to configure.
2. Select **myrealm**.

   The settings for **myrealm** appear.
3. Click the **Providers** tab.

   The list of authentication providers appears.
4. Click the **DefaultIdentityAsserter** provider in the table.

   The settings for **DefaultIdentityAsserter** appear.
5. Click the **Common** tab.
6. Under **Active types**, move `X.509` from the **Available** list to the **Chosen** list if it is not there already.
7. Click the **Provider Specific** tab.
8. Set the **Default User Name Mapper Attribute Type** is to `CN` if it is not set to `CN` already.

To configure the keystore:

1. Select **Environment** from the **Domain Structure** menu.

   A summary of the environment appears.

2. Select **Servers**.

   A summary of servers appears.

3. Click **AdminServer.**

   The settings for the AdminServer appear.

4. Click the **Keystores** tab. The keystore settings appear.

5. Configure the Identity and Trust sections of the keystore form. See "Configuring Identity and Trust" in *Oracle WebLogic Server Administering Security for Oracle WebLogic Server* for information about setting the keystore values.

6. Click **Save** to save your configuration.

Removing Outbound Web Security

To turn off outbound security associated with a particular WS-Policy file, you must edit the **plan.xml** file that is created when you attach Policy to a web service, as in step 8 above. The default location is: */domain_home/wlng-access-network-domain***/servers/AT1/stage/wlng_at/plan/Plan.xml**, but your location may be different. Make sure the *value* element is set to `inbound` as in the following example:

```
<variable>
      <name>WsPolicy_policy:Auth.xml_Direction_11745107731400</name>
      <value>inbound</value>
</variable>
```

Removing WS-Policy from a Web Service

Services Gatekeeper 2.2 and earlier use a different mode of authentication than the WS-Security model. Services Gatekeeper can be configured to support applications designed to work using the older model, but the WS-Policy that is configured by default must be removed.

> **Note:** The easiest way to do this is to make these changes *before* you start Services Gatekeeper for the first time. Certain configuration values are cached on start up.
>
> For example, if you started Services Gatekeeper during the post-install phase to set up additional JMS Servers, you must redeploy the `wlng_at.ear` file *after* you have made your changes.

To remove the policy files from a web service:

1. Expand the EAR file for the access tier part of the communication service.
2. Expand the war file for the web service in question.
3. Modify the weblogic-webservices-policy.xml file for that web service to remove the policy entries.

   Example 5–2 shows the `weblogic-webservices-policy.xml` file for a web service with policy entries.

Example 5–2 weblogic-webservices-policy.xml with policy entries

```
<?xml version='1.0' encoding='UTF-8'?>
<webservice-policy-ref xmlns="http://www.bea.com/ns/weblogic/90"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<port-policy>
  <port-name>AudioCall</port-name>
  <ws-policy>
   <uri>policy:Auth.xml</uri>
   <direction>inbound</direction>
  </ws-policy>
</port-policy>
</webservice-policy-ref>
```

Example 5–3 shows the same file for that web service after the policy entries were removed.

Example 5–3 weblogic-webservices-policy.xml with policy entries removed

```
<?xml version='1.0' encoding='UTF-8'?>
<webservice-policy-ref xmlns="http://www.bea.com/ns/weblogic/90"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"></webservice-policy-ref>
```

4. Re-package the war file with the modified `weblogic-webservices-policy.xml` file
5. Re-package the ear file.
6. Copy the modified ear file to the domain directory of the administration server.
7. If you have previously run Services Gatekeeper, you should now re-deploy the EAR file using the Administration Console.

Creating and Using a custom WS-Policy

For information about creating and using a custom policy file for message-level security, see "*Creating the Web Service Reliable Messaging WS-Policy File*" in *Oracle WebLogic Server Developing JAX-WS Web Services for Web Services* here:

`http://www.oracle.com/pls/topic/lookup?ctx=wsc70&id=WSGET290`

Also see the *Oracle WebLogic Server Administration Console Online Help*.

Available default WS-Policies

WS-Policy files can be used to require applications clients to authenticate, digitally encrypt, or digitally sign SOAP messages. By default, Services Gatekeeper supplies these policy files: **auth.xml**, **encrypt.xml**, **sign.xml**, and **UsernameTokenDigestPolicy.xml**. If the built-in WS-Policy files do not meet your security needs, you can build custom policies.

WS-Policy assertions are used to specify a web service requirement for digital signatures and encryption, along with the security algorithms and authentication mechanisms that it requires; for example, Policy for SAML.

Using SAML Tokens Between RESTful Applications

Services Gatekeeper supports using SAML tokens to pass authentication and authorization between REST-based applications. The most common use for SAML is creating single sign-on (SSO) logins. Services Gatekeeper supports allowing users to provide credentials once, and then obtain access to a group of related REST-based applications. This streamlines the user experience by allowing them to enter credentials once and then access multiple applications.

Services Gatekeeper allows the use of SAML tokens among REST-based applications that interact with the access tier. It is not supported by the Partner and API Management Portal.

See "Supported SAML Specification" in *Statement of Compliance* for the SAML specification that this release of Services Gatekeeper supports.

Securing RESTful Web Services with SSL

Services Gatekeeper uses SSL authentication to secure REST-based Web services. You have the options listed in Table 5–1 for security levels.

Table 5–1 SSL Security Options

SSL Level	Client Response Setting	Notes
One-Way	Client certificates not required	Essentially no security. Only use in a test and evaluation implementation.
Two-Way	Client certificates requested but *not* enforced.	The default setting. Appropriate for a test and evaluation system. A client certificate is requested but if one is not returned, the session continues normally.
Two-Way	Client certificates requested and enforced.	The most secure setting; appropriate for most production implementations. A client certificate is requested and if one is not returned the session is terminated.

Table 5–2 lists the default keystore and truststore name and locations. If you are configuring a test and evaluation implementation it is acceptable to use these values. If you are configuring a production implementation, replace these values with settings that you create.

Table 5–2 Default SSL Truststore and Keystore Values

Default Credential	Default Value
Demo identity keystore location	*Oracle_home*/**Middleware/wlserver_10.3/server/lib/DemoIdentity.jks**
Demo identity keystore password	**DemoIdentityKeyStorePassPhrase**
Demo identity keystore (private key) password	**DemoIdentityKeyStorePassPhrase**

Table 5–2 (Cont.) Default SSL Truststore and Keystore Values

Default Credential	Default Value
Default trust keystore location	*Oracle_home*/**Middleware/wlserver_10.3/server/lib/DemoTrust.jks**
Default trust keystore password	**DemoTrustKeyStorePassPhrase**

Configuring Application-Facing Servers for SSL

Follow the instructions in the following sections to set up and configure SSL security for RESTful communication between Services Gatekeeper and external applications:

- Enabling and Configuring SSL for Each Application Tier Server
- Adding Certificates to the Application Tier Servers and Applications

Enabling and Configuring SSL for Each Application Tier Server

To enable SSL, perform these steps on each of your application tiers that use RESTful communication:

1. Open the WebLogic Server Administration Console.
2. Click **Lock and Edit**.
3. Navigate to **Environment**, then **Servers**, then *AT_server_name*, then the **Configuration** tab, then the **General** subtab.
4. Check the **SSL Listen Port Enabled** check box.
5. Set the SSL Listen Port: number to **7002**.
6. Navigate to the **Keystores** subtab.
7. If you are setting up a production environment, change these settings to values that you create. Use the default values (listed in Table 5–2) for test and evaluation implementations:
   a. **Keystores**: Click Change.
   b. **Demo Identity Keystore**: Change this to different location that you have created.
   c. **Demo Identity Keystore Passphrase**: Change this to a secure password that you create.
   d. **Demo Trust Keystore**: Change this to a location that you have created.
   e. **Demo Trust KeyStore Passphrase**: Change this to a secure password that you create.
8. Navigate to the **SSL** tab.
9. If you are setting up a production environment, change these settings to values that you create. Use the default values (listed in Table 5–2) for test and evaluation implementations:
   a. **Identity and Trust Location:** Click **Change**.
   b. **Private key Passphrase**: Change this to a secure password that you choose.
   c. **Two Way Client Cert Behavior:** See Table 5–1 for the possible values. Select the value appropriate for your implementation.
10. Click **Release Configuration.**

Adding Certificates to the Application Tier Servers and Applications

Once you have configured SSL and key- and truststores for each application-facing server, you must create a file of trust certificates to import into the client keystore and truststore locations.

To do so:

1. Open a shell on the Services Gatekeeper server.
2. Change the directory to the location of your trust store. You set this with the directions in "Enabling and Configuring SSL for Each Application Tier Server". The default directory is *Oracle_home*/**Middleware/wlserver/server/lib**.
3. (Optional) List the credentials to confirm that they are correct with the following command. The default values for keystore_name and:

   ```
   keytool -list -v -storetype JKS -keystore keystore_name.jks -storepass
   keystore_password
   ```

4. Export the certificates from the Services Gatekeeper keystore into a file with this command:

   ```
   keytool -exportcert -v -storetype JKS -keystore keystore_file.jks -storepass
   keystore_password -alias nt1-cacert -rfc -file SG_trust_cacert_file.cer
   ```

   Where:

   keystore_file.**jks** is the Services Gatekeeper keystore file you set in the "Enabling and Configuring SSL for Each Application Tier Server" section.

   keystore_password is the keystore password you created in the "Enabling and Configuring SSL for Each Application Tier Server" section.

   SG_trust_cacert_file.**cer** is the Services Gatekeeper truststore certificate file.

   This command exports the keystores into a file called *export_file*.**cer** that you use to import certificates into the Services Gatekeeper trust store.

5. Import *SG_trust_cacert_file*.**cer** into the client application's keystore. See your application product documentation for details.

   Note: Do this for each server communicating with Services Gatekeeper.

6. Obtain your client application's truststore certificate file. See your client application or client server documentation for details. You need the truststore certificate file with the **.cer** file extension.
7. Change the directory to the location of your keystore. You set this with in the "Enabling and Configuring SSL for Each Application Tier Server" section. The default directory is *Oracle_home*/**Middleware/wlserver**_*release_level*/**server/lib**
8. Import your client application's certificates into the Service Gatekeeper truststore with this command:

   ```
   keytool -importcert -v -alias client-cacert -file client_cacert_file.cer
   -storetype JKS -keystore truststore_file.jks  -storepass truststore_password
   ```

   Where:

   client_cacert_file.**cer** is the client truststore certificates file.

truststore_file.**jks** is the Services Gatekeeper truststore file you set in the "Enabling and Configuring SSL for Each Application Tier Server" section.

truststore_password is the truststore password you created in the "Enabling and Configuring SSL for Each Application Tier Server" section.

Securing Network-Facing Servers With Keystores

To secure RESTful Web service traffic with network-facing servers, you must:

- Import the network node server certificates into Services Gatekeeper keystore as trusted entries,
- Export your Services Gatekeeper keystores into the network node server.

To secure RESTful traffic, you must first know the location of the network server certificate file (*keystore_cacert_file*.**cer**) that you created in the "Adding Certificates to the Application Tier Servers and Applications" section.

To secure network-facing traffic:

1. Open a shell on the Services Gatekeeper server.
2. Change directory to the location of your Services Gatekeeper keystore. You set this in the "Enabling and Configuring SSL for Each Application Tier Server" section. The default directory is *Oracle_home*/**Middleware/wlserver/server/lib**.
3. To import the network server keystore credentials, generate a public/private key pair, and create a keystore file, use this command:

```
keytool -genkeypair -v -alias key_pair -keyalg RSA -storetype JKS
-validity 3650 -keystore keystore_file.jks -dname "domain_name"  -storepass
public_key_password -keypass private_key_password
```

   Where:

   key_pair is the name of the public/private key pair.

   public_key_password is the public key for the public/private key pair.

   keystore_file.**jks** is the Services Gatekeeper Java keystore file accepting the keystore credentials that is created with this command.

   domain_name is the distinguished name of the network node credential key, for example: “CN=oracle-wac-gw-nt1,OU=CGBU,O=Oracle,L=ANY,C=US”.

   private_key_password is the private key of the public/private key pair.

4. To import the Services Gatekeeper keystore credentials into the network node server, use this command:

```
keytool -exportcert -v -alias key_pair -keystore keystore_file.jks
-storetype JKS -storepass public_key_password -rfc -file cacert_file.cer
```

   Where:

   key_pair is the name of the public/private key pair.

   keystore_file.**jks** is the Services Gatekeeper keystore file to import.

   public_key_password is the public key password for the certificates.

   cacert_file.**cer** is the certificate file that is created by this command.

5. Start the Administration Console (or another MBean browser) and navigate to the **MBeans** tab.

6. Click **Lock and Edit.**
7. Check the **Display: tree** check box.
8. Navigate to **wlng**, then **RESTfulClientSSL**.
9. Edit these fields to direct Services Gatekeeper to use the keystores you have imported:
   - **KeyStoreFile**: Enter the location of the *keystore_file***.jks** file you imported in step 3.
   - **KeyStorePassPhrase**: Enter the *publk_key_password* you used in step 3.
   - **KeyStoreType:**: Enter **JKS**.
   - **PrivateKeyPhrase:**: Enter the *private_key_password* that you created in step 3.
10. Click **Save**.
11. Click **Activate Changes**.

Securing the Oracle Access Manager MBeans

The Java Management Extension (JMX) OAM MBeans control access to the Services Gatekeeper OAM functionality. You can change these MBean settings using the Administration Console or through an external mechanism. Access to these MBeans are controlled by JMX Policy, which associates administrative user groups with access privilege levels. By default any administrative user can access and change the Services Gatekeeper OAM MBeans. You may want use the instructions in this section to add more restrictive access control to prevent inadvertent or malicious changes to these MBeans. However, the decision should be based on your implementation's security policies.

Administrative Groups

Administrative users and groups are set up as described in the "Managing Users and User Groups" section. To control how these users have access to MBeans, and thus OAM functionality, you must assign JMX Policy to these user groups. You use WebLogic Server Administration Console to do this. For details see the discussion on managing security policies in *Oracle WebLogic Server Administration Console Online Help* at:

`https://docs.oracle.com/cd/E24329_01/apirefs.1211/e24401/taskhelp/security/ManageSecurityPolicies.html`

Each policy can do the following:

- Control read access for all an MBean's attributes or for specific attributes that you select.
- Control write access for all an MBean's attributes or for specific attributes that you select.
- Control invoke access for all an MBean's operations or for specific operations that you select.

For example, to give a user complete access to an MBean, select the WLNG_Administrator's Group in the policy condition.

Administrative Service Groups

In addition to controlling access to OAM functionality in a general way – ReadOnly, ReadWrite, and so on – you may also want to control access by Service group. So, for example, if you have users whose job is limited to setting up and managing Application Service Providers through a system using the Partner Relationship Management interfaces, you might want to give them, and only them, ReadWrite privileges, but only to a subset of the available MBeans, those having to do with the operator part of those transactions. To do this, you have to create custom XACML policies to attach to these subsets. Services Gatekeeper uses the standard WebLogic Server mechanisms for this purpose. For the basic process you must:

- Determine the special identifier (called the resourceId) for each MBean.
- Create an XACML policy for a security role.
- Specify one or more Rule elements that define which users, groups, or roles belong to the new security role.
- Attach this role to the MBean by way of the resourceId.

For details on using XACML documents to secure WebLogic resources, see "Using XACML Documents to Secure WebLogic Resources" in *Oracle WebLogic Server Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

For a reference for XACML on WebLogic Server, see “Reference for XACML on WebLogic Server” in *Oracle WebLogic Server Securing Resources Using Roles and Policies for Oracle WebLogic Server.*

6

Setting Up Geographic Redundancy

This chapter describes how to set up geographically redundant site sets for Oracle Communications Services Gatekeeper. The attributes and operations supporting geographic redundancy are explained, and a configuration workflow is provided.

Understanding Geographic Redundancy

The Geographic Redundancy service replicates data between geographically distant sites so that applications can switch to a different site to process traffic (for example, in case of the catastrophic failure of one). All geographically-redundant sites include all configuration data (account information, system Service Level Agreements (SLAs), and budgets) necessary for SLA enforcement available. For more overview information about geographic redundancy, see “Redundancy, Notifications, Load Balancing, and High Availability” in *Services Gatekeeper Concepts*.

Each set of geographically redundant sites has a single *geomaster* system, which is connected to a *slave* system or systems. These sites are frequently set up in geomaster-slave pairs, but a single geomaster can have any number of slaves. Each geographic site has a name which is used for looking up data relevant to the site systems. The names of the slaves sites are defined in the geomaster system. Depending on how you configure geographic redundancy, each application may have to configure communication with all sites, or just one. Subsequently, the application needs to register for subscriptions on one site only. During a site failure, the other site automatically handles additional subscription messages.

Understanding the Geographic Redundancy Configuration Options

You have these options for setting up geographic redundancy:

- Configuring basic geographic redundancy so that all applications must register with each site. This option is most practical if you do not have too many applications. This option requires the most up-front configuration, but requires less database storage space. See "Configuring Basic Geographic Redundancy" for instructions.
- Configuring geographic redundancy so that applications need only register with one site. This option is most practical if you have a lot of applications (or sites) to administer. This option requires less up-front configuration, but requires more database space. See "Configuring Geographic Redundancy Without Registering Applications at Every Site" for instructions.

  This style of geographic redundancy is only available on a MySQL database and for the following types of SMS, MMS, or terminal location traffic:

- SMS
  - ParlayX
  - OneAPI
  - Native SMPP
- MMS
  - ParlayX
  - OneAPI
  - Native MM7
- Terminal Location
  - ParlayX

About Recovering from a Geographic Redundant Failure

If a geographically redundant domain fails:

- An application's connection and login time-out or receive an error response from domain load balancer.
- The application then logs in to the peer geographically redundant domain (if not already logged in) and resumes sending and receiving traffic.
- SLA and policy requests, and budget limits optionally continue to be enforced in peer domain. You control this behavior with configuration settings.
- Applications should periodically attempt to login to the failed domain.
- When the failed domain recovers, the application logs back into the original domain, and logs out of the peer.
- Alternatively, an application could continue to use the peer domain until reset by the operator.

Configuring Basic Geographic Redundancy

These tasks are required to configure basic geographic redundancy. You must do all of them at each site:

- "Configuring Both Sites for Geographic Redundancy". You must do this at each site.
- "Defining One Site as the GeoMaster". You must do this at each site.

Configuring Both Sites for Geographic Redundancy

To use geographic redundancy, each site must be appropriately configured using **GeoRedundantServiceMBean**. Using this service, you:

- Define the ID of the local site in the **GeoSiteId** field.
- Define the number of failed attempts to reach a remote site before an alarm should be raised in the **RemoteSiteReachabilityAlarmThreshold** field.
- Define the remote site using the **setSiteAddress** method.

See "GeoRedundantService Reference" for details.

Defining One Site as the GeoMaster

One site of the site pair must be designated the **geomaster** site using the **GeoStorageServiceMBean**. You define geomaster site using the **GeoMasterSiteId** field

See "GeoStorageService Reference" for details.

Configuring Geographic Redundancy Without Registering Applications at Every Site

This sections explains how to configure geographic redundancy so that applications are only required to register for notifications at a single geographically redundant site. Delivery receipts that are delivered to any site and are replicated across all sites.

Each geographically dispersed site hosts a Services Gatekeeper domain (collection of managed servers and clusters). You deploy and manage these geographically redundant sites independently. It is possible to distribute load across these domains. That is, multiple active geographically redundant domains are supported so that applications may connect to any of the geographically redundant domains. These geographically redundant domains synchronize traffic data, and enforce SLAs and Policies by periodically synchronizing traffic data between each other. The interval between these synchronizations is based on a configurable synchronization interval.

The geographically redundant implementations:

- Are deployed and managed independently
- Perform bi-directional data synchronization between geographic domain pairs
- Have failover monitoring mechanisms, and issue fire alarms when they detect problems

You configure this geographically redundant option by replicating database tables that contain messaging information among the geographically redundant sites, and use the WebLogic write-through coherence mode. Once these tasks are complete, your applications need only apply for subscriptions from one of the geographically redundant sites.

Understanding the Configuration Workflow

To set up geographic redundancy so that applications do not have to register with all sites, follow the steps in these sections:

- Understanding the Configuration Prequisites
- Understanding the Configuration Example
- Configuring the MySQL Database
- Configuring the Services Gatekeeper Cache
- Configuring Services Gatekeeper MBeans
- Configuring Cache Types
- Protecting the Modified MBean Files
- Changing the type_id of Tables Replicated by MySQL
- Merging the Data Tables
- Restarting the MySQL Servers

- Connecting the Master and Slave Databases

Understanding the Configuration Prequisites

Before starting the configuration process:

- Services Gatekeeper must be installed on all of the hosts.
- A MySQL database must be installed on all of the hosts

  The Services Gatekeeper domains must be installed on all of the hosts.

  You followed the instruction in "Configuring Basic Geographic Redundancy" and configured basic geographic redundancy.

Understanding the Configuration Example

The configuration instructions in this section use the example systems listed here. Both of these systems have Services Gatekeeper and MySQL 5.6 installed, and have been configured as master/slaves in a geographically redundant cluster:

- **geotest1**, with an IP address of 10.161.159.189
- **geotest2** with an IP address of 10.161.159.166

This guide uses the **geotest1** and **geotest2** hostnames in the configuration steps for clarity; however it is best practice guideline to use IP addresses in a production system.

Table 6–1 lists values that are used in the configuration example.

Table 6–1 Geographically Redundant Configuration Elements

Configuration Element	geotest1 Value	geotest2 Value
db_name	ocsg60_geosite1	ocsg60_geosite2
GeoRedundantService.GeoSiteId	site1	site2
GeoRedundantService.getSiteAddress(...)	(site2) ->t3://10.161.159.166:9001, 10.161.159.166:9101	(site1) -> t3://10.161.159.189:9001,10.161.159.189:9101
GeoStorageService.GeoMasterSiteId	site1	site2
AccountService.SessionRequired	Disabled	Disabled
SMPPServiceMBean.ConnectionBasedRouting	False	False
SMPPServiceMBean.OfflineMO	True	True
SMPPServiceMBean.SkipAddressrangeCheckInBindRequest	True	True

Configuring the MySQL Database

Configure the MySQL servers on your master/slave nodes as both masters and slaves. You do this by using the **my.cnf** file. The example **my.cnf** files in this section shows you how.

Example 6–1 lists the **my.cnf** configuration setting used by **geotest1**:

Example 6–1 Example geotest1 my.cnf file

```
[mysqld]
server-id              = 1
log_bin                = /var/lib/mysql/mysql-bin.log
#This is the database to log statements for
binlog-do-db           = ocsg60_geosite1

#The remote site uses a different name, need to map it to our local name
replicate-rewrite-db   = ocsg60_geosite2->ocsg60_geosite1

##Tables to replicate (local db name, ie after rewrite)
#SMS
replicate-do-table     = ocsg60_geositea.native_smpp_session_store
replicate-do-table     = ocsg60_geositea.pl_sms_delivery_notif
replicate-do-table     = ocsg60_geositea.pl_sms_offline_notif
replicate-do-table     = ocsg60_geositea.pl_sms_online_notif
replicate-do-table     = ocsg60_geositea.pl_sms_smpp_mt_sms
replicate-do-table     = ocsg60_geositea.pl_sms_smpp_mt_dr
replicate-do-table     = ocsg60_geositea.pl_sms_smpp_mo_sms


#MMS
replicate-do-table     = ocsg60_geositea.pl_mms_mt_dr_mms
replicate-do-table     = ocsg60_geositea.pl_mms_mo_mms
replicate-do-table     = ocsg60_geositea.pl_mms_mo_content_mms
replicate-do-table     = ocsg60_geositea.pl_legacy_mms_mt
replicate-do-table     = ocsg60_geositea.pl_mms_dr_subscribe
replicate-do-table     = ocsg60_geositea.pl_mms_offline_notif
replicate-do-table     = ocsg60_geositea.pl_mms_online_notif
replicate-do-table     = ocsg60_geositea.pl_legacy_mm7_notif
replicate-do-table     = ocsg60_geositea.pl_legacy_mms_status
replicate-do-table     = ocsg60_geositea.pl_legacy_mms_vas_id
replicate-do-table     = ocsg60_geositea.pl_legacy_mms_vasp_id

#TL
replicate-do-table     = ocsg60_geositea.pl_tl_mlp_trigger_info
```

Example 6–2 lists the **my.cnf** configuration file used by **geotest2**:

Example 6–2 Example geotest2 my.cnf File

```
[mysqld]
server-id              = 2
log_bin                = /var/lib/mysql/mysql-bin.log

#The database to log statements for
binlog-do-db           = ocsg60_geosite2

#The remote site uses a different name, need to map it to our local name
replicate-rewrite-db   = ocsg60_geosite1->ocsg60_geosite2

##Tables to replicate (local db name, ie after rewrite)
#SMS
replicate-do-table     = ocsg60_geositeb.native_smpp_session_store
replicate-do-table     = ocsg60_geositeb.pl_sms_delivery_notif
replicate-do-table     = ocsg60_geositeb.pl_sms_offline_notif
replicate-do-table     = ocsg60_geositeb.pl_sms_online_notif
replicate-do-table     = ocsg60_geositeb.pl_sms_smpp_mt_sms
replicate-do-table     = ocsg60_geositeb.pl_sms_smpp_mt_dr
replicate-do-table     = ocsg60_geositeb.pl_sms_smpp_mo_sms
```

```
#MMS
replicate-do-table     = ocsg60_geositeb.pl_mms_mt_dr_mms
replicate-do-table     = ocsg60_geositeb.pl_mms_mo_mms
replicate-do-table     = ocsg60_geositeb.pl_mms_mo_content_mms
replicate-do-table     = ocsg60_geositeb.pl_legacy_mms_mt
replicate-do-table     = ocsg60_geositeb.pl_mms_dr_subscribe
replicate-do-table     = ocsg60_geositeb.pl_mms_offline_notif
replicate-do-table     = ocsg60_geositeb.pl_mms_online_notif
replicate-do-table     = ocsg60_geositeb.pl_legacy_mm7_notif
replicate-do-table     = ocsg60_geositeb.pl_legacy_mms_status
replicate-do-table     = ocsg60_geositeb.pl_legacy_mms_vas_id
replicate-do-table     = ocsg60_geositeb.pl_legacy_mms_vasp_id

#TL
replicate-do-table     = ocsg60_geositeb.pl_tl_mlp_trigger_info
```

Configuring the Services Gatekeeper Cache

Change the relevant Services Gatekeeper cache configuration so that data is read and written directly to the database instead of being cached.

This server-cache configuration guarantees that for example, when you send an SMS message, the correlation data is available to both geographically-redundant sites when the delivery receipt is received. There is a trade off in performance.

Replicate the following database tables for geographic redundancy, both in MySQL and in the Services Gatekeeper cache:

- native_smpp_session_store
- pl_legacy_mm7_notif
- pl_legacy_mms_mt
- pl_legacy_mms_status
- pl_legacy_mms_vas_id
- pl_legacy_mms_vasp_id
- pl_mms_dr_subscribe
- pl_mms_mo_content_mms
- pl_mms_mo_mms
- pl_mms_mt_dr_mms
- pl_mms_offline_notif
- pl_mms_online_notif
- pl_sms_delivery_notif
- pl_sms_offline_notif
- pl_sms_online_notif
- pl_sms_smpp_mo_sms
- pl_sms_smpp_mt_dr
- pl_sms_smpp_mt_sms

- pl_tl_mlp_trigger_info

Replicating SMS Configuration Database Tables

To replicate SMS messaging database tables across geographically redundant sites:

1. Navigate to *domain_home*/**config/store_schema** of your administration server.
2. Expand the **com.bea.wlcp.wlng.plugin.sms.common.store_***release***.jar** file.
3. Open the **wlng-cachestore-config-extensions.xml** file for editing.
4. Make these changes for the type of application you are using:
   - OneAPI Applications:

     Change: **wlng.db.wt.plugin.sms.common.sms_delivery_notif_store**

     To: **geo.wlng.local.wt.plugin.sms.common.sms_delivery_notif_store**
   - OneAPI and Parlay X applications:

     Change: **wlng.db.wt.plugin.sms.common.sms_offline_notif_store**

     To: **geo.wlng.local.wt.plugin.sms.common.sms_offline_notif_store**

     And

     Change: **wlng.dg.wt.plugin.sms.common.sms_online_notif_store**

     To: **geo.wlng.local.wt.plugin.sms.common.sms_online_notif_store**
5. Save and close **wlng-cachestore-config-extensions.xml** file.
6. If you use Extended Web Service or Binary SMS applications, expand the **oracle.ocsg.plugin.sms.px21.smpp.store_***release***.jar** file.
   - Change: **wlng.db.wt.plugin.sms.common.sms_binary_notif_store**
   - To: **geo.wlng.local.wt.plugin.sms.common.sms_binary_notif_store**
7. Stop, and restart all NT servers in your Services Gatekeeper implementation.

Replicating MMS Configuration Database Tables

To replicate MMS messaging database tables across geographically redundant sites:

1. Navigate to *domain_home*/*domain_name*/**config/store_schema**.
2. For OneAPI applications, expand the **com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_***release***.jar** file.
3. Open the **wlng-cachestore-config-extensions.xml** file for editing.
4. Make these changes:
   - Change: **wlng.db.wt.plugin.mms.common.mms_dr_subscribe_store**
   - To: **geo.wlng.local.wt.plugin.mms.common.mms_dr_subscribe_store**
5. Save and close the file.
6. For OneAPI and Parlay X 2.1 applications, expand the **com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_***release***.jar** file.
7. Open the **wlng-cachestore-config-extensions.xml** file for editing.
8. Make these changes:
   - Change **wlng.db.wt.plugin.mms.common.mms_offline_notif_store**

- To: **geo.wlng.local.wt.plugin.mms.common.mms_offline_notif_store**

and

- Change: **wlng.db.wt.plugin.mms.common.mms_online_notif_store**
- To: **geo.wlng.local.wt.plugin.mms.common.mms_online_notif_store**

9. Save and close the file.
10. Stop, and restart all NT servers in your Services Gatekeeper implementation.

Replicating Terminal Location Configuration Database Tables

The terminal location **pl_tl_mlp_trigger_info** database table stores both user and traffic information which make it impractical to replicate using the Service Gatekeeper geographical redundancy feature. Instead, use the MySQL database table replication feature to replicate this table. See your MySQL documentation for details. If you use a different database, see that product documentation for instructions on how to replicate database tables.

Configuring Services Gatekeeper MBeans

See Table 6–1 for the list of MBeans and the settings required.

Configuring Cache Types

Manually modify these storage schema jar files for geographic redundancy:

- oracle.ocsg.plugin.sms.native.smpp.store_6.0.0.0.jar
- com.bea.wlcp.wlng.plugin.multimediamessaging.mm7.store_6.0.0.0.jar
- com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_6.0.0.0.jar
- com.bea.wlcp.wlng.plugin.sms.common.store_6.0.0.0.jar
- com.bea.wlcp.wlng.plugin.terminallocation.mlp.store_6.0.0.0.jar
- oracle.ocsg.plugin.sms.px21.smpp.store_6.0.0.0.jar

Protecting the Modified MBean Files

In order to keep subsequent patches from overwriting the jar files you modified in "Configuring Cache Types", protect them with these steps:

Run these commands on each file in sequence:

```
% jar xf store_schema_filename.jar wlng-cachestore-config-extensions.xml
% mv wlng-cachestore-config-extensions.xml store_schema_filename.xml
% chmod 640 chmod 640 store_schema_filename.xml.xml
```

For example:

```
% jar xf com.bea.wlcp.wlng.plugin.multimediamessaging.mm7.store_6.0.0.0.jar
wlng-cachestore-config-extensions.xml
% mv wlng-cachestore-config-extensions.xml
com.bea.wlcp.wlng.plugin.multimediamessaging.mm7.store_6.0.0.0.xml
% chmod 640 chmod 640 com.bea.wlcp.wlng.plugin.multimediamessaging.mm7.store_
6.0.0.0.xml
```

Tip: Modify these files in your administration server then copy them to each managed server to ensure that each server has identical configuration.

Note: When changing cache configuration you MUST do a full cluster restart (NOT a rolling restart) to avoid having nodes with conflicting cache configuration. Follow these steps:

1. Stop all nodes in the cluster.
2. Start the administration server.
3. Start the rest of the servers in the domain.

Changing the type_id of Tables Replicated by MySQL

For each table that is replicated by MySQL, you must change the **type_id** (**cache_name** for **native_smpp_session_store** table) to match the value for **wlng.db.direct** in the **wlng-cachestore-config-extensions.xml** file.

This example changes the **type_id** for the **pl_legacy_mm7_notif** table:

1. Open **com.bea.wlcp.wlng.plugin.multimediamessaging.mm7.store_6.0.0.0.xml** for editing.
2. Search for the table name (**pl_legacy_mm7_notif**).
3. Change the value for **type_id** from **wlng.db.wt.plugin.mms.legacy.mm7.mo_notif_store** to **wlng.db.direct.plugin.mms.legacy.mm7.mo_notif_store**.

Table 6–2 lists the table names and type_ids.

Table 6–2 type_id values for MySQL Tables

File	Table Name	Type ID
oracle.ocsg.plugin.sms.native.smpp.store_6.0.0.0.xml	native_smpp_session_store	wlng.db.direct.native.smpp.sessionInfo.store
oracle.ocsg.plugin.sms.native.smpp.store_6.0.0.0.xml	pl_legacy_mm7_notif	wlng.db.direct.plugin.mms.legacy.mm7.mo_notif_store
oracle.ocsg.plugin.sms.native.smpp.store_6.0.0.0.xml	pl_legacy_mms_mt	wlng.db.direct.plugin.mms.legacy.mm7.mt_mm7_state
oracle.ocsg.plugin.sms.native.smpp.store_6.0.0.0.xml	pl_legacy_mms_status	wlng.db.direct.plugin.mms.legacy.mm7.mo_status_report
oracle.ocsg.plugin.sms.native.smpp.store_6.0.0.0.xml	pl_legacy_mms_vas_id	wlng.db.direct.plugin.mms.legacy.mm7.mt_mm7_vas_id
oracle.ocsg.plugin.sms.native.smpp.store_6.0.0.0.xml	pl_legacy_mms_vasp_id	wlng.db.direct.plugin.mms.legacy.mm7.mt_mm7_vasp_id
com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_6.0.0.0.xml	pl_mms_dr_subscribe	wlng.db.direct.plugin.mms.common.mms_dr_subscribe_store
com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_6.0.0.0.xml	pl_mms_mo_content_mms	wlng.db.direct.plugin.mms.common.content.mo_mms_store
com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_6.0.0.0.xml	pl_mms_mo_mms	wlng.db.direct.plugin.mms.common.mo_mms_store

Table 6–2 (Cont.) type_id values for MySQL Tables

File	Table Name	Type ID
com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_6.0.0.0.xml	pl_mms_mt_dr_mms	wlng.db.direct.plugin.mms.common.deliveryreport.mt_mms_store
com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_6.0.0.0.xml	pl_mms_offline_notif	wlng.db.direct.plugin.mms.common.mms_offline_notif_store
com.bea.wlcp.wlng.plugin.multimediamessaging.px21.mm7.store_6.0.0.0.xml	pl_mms_online_notif	wlng.db.direct.plugin.mms.common.mms_online_notif_store
com.bea.wlcp.wlng.plugin.sms.common.store_6.0.0.0.xml	pl_sms_delivery_notif	wlng.db.direct.plugin.sms.smpp.delivery_notif_store
com.bea.wlcp.wlng.plugin.sms.common.store_6.0.0.0.xml	pl_sms_offline_notif	wlng.db.direct.plugin.sms.common.sms_offline_notif_store
com.bea.wlcp.wlng.plugin.sms.common.store_6.0.0.0.xml	pl_sms_online_notif	wlng.db.direct.plugin.sms.common.sms_online_notif_store
oracle.ocsg.plugin.sms.px21.smpp.store_6.0.0.0.jar	pl_sms_smpp_mo_sms	wlng.db.direct.plugin.sms.smpp.mo_sms_store
oracle.ocsg.plugin.sms.px21.smpp.store_6.0.0.0.jar	pl_sms_smpp_mt_dr	wlng.db.direct.tc.plugin.sms.smpp.mt_dr_store
oracle.ocsg.plugin.sms.px21.smpp.store_6.0.0.0.jar	pl_sms_smpp_mt_sms	wlng.db.wb.direct.plugin.sms.smpp.mt_sms_store
com.bea.wlcp.wlng.plugin.terminallocation.mlp.store_6.0.0.0.xml	pl_tl_mlp_trigger_info	wlng.db.direct.plugin.tl.mlp.trigger_info

Configuring the my.cnf File for MySQL Replication

Table 6–3 lists the **my.cnf** file configuration settings to configure two systems as a geographically redundant pair. This table continues using the geotest1 and geotest2 example systems.

Table 6–3 my.cnf Configuration Entries

Entry	geotest1 my.cnf File Value	geotest2 my.cnf File Value	Description
bind-address	Comment out	Comment out	This configuration requires that all servers listen to an IP address that is reachable by all peers. Commenting out this line directs the system to listen to all interfaces.
server-id	server-id=1	server-id=2	This assigns MySQL server names to our example systems.
binlog-do-db	ocsg60_geosite1	ocsg60_geosite2	Each site is both a master and slave. This entry directs each master to log database statements to themselves.
log_bin	/var/lib/mysql/mysql-bin.log	/var/lib/mysql /mysql-bin.log	Specifies the bin log location. Use the same value for both systems.

Table 6–3 (Cont.) my.cnf Configuration Entries

Entry	geotest1 my.cnf File Value	geotest2 my.cnf File Value	Description
replicate-rewrite-db	ocsg60_geosite2->ocsg60_geosite1	ocsg60_geosite1->ocsg60_geosite2	This entry directs the slave domain to receive statements from the master domain. You can omit this entry if you use the same database name on both systems.
replicate-do-table	See Example 6–1 for the list of tables to replicate.	See Example 6–2 for a list of the tables to replicate.	NA

Merging the Data Tables

Once you have merged the tables, avoid writing any data to them until you have completed configuration.

Restarting the MySQL Servers

Restart both MySQL servers to make you changes take effect with this command:

```
sudo service mysql restart
```

Connecting the Master and Slave Databases

You must connect the databases in your geographically redundant systems with the instructions in this section:

1. Create a user called **replicator** on each system to do the replication, and grant them rights with these commands:

   ```
   mysql> create user 'replicator'@'%' identified by 'password';
   mysql> grant replication slave on *.* to 'replicator'@'%';
   ```

   See the MySQL **create user** and **grant** commands for details.

2. Obtain each master's File and Position to connect to. Run these commands on each system:

   ```
   mysql -u root -p -h geotest1
   mysql> show master status
   ```

   Record the **File** and **Position** for each system. The output will look something like this:

   geotest1:

   ```
   +------------------+----------+-----------------+------------------+
   | File             | Position | Binlog_Do_DB    | Binlog_Ignore_DB |
   +------------------+----------+-----------------+------------------+
   | mysql-bin.000002 | 85527658 | ocsg60_geositea |                  |
   +------------------+----------+-----------------+------------------+
   1 row in set (0.03 sec)
   ```

   geotest2:

   ```
   +------------------+----------+-----------------+------------------+
   | File             | Position | Binlog_Do_DB    | Binlog_Ignore_DB |
   +------------------+----------+-----------------+------------------+
   | mysql-bin.000006 | 12927236 | ocsg60_geositeb |                  |
   ```

```
+------------------+----------+----------------+------------------+
1 row in set (0.07 sec)
```

3. Run these commands on all systems to connect slaves to the masters:

   On geotest1:

   ```
   mysql -u root -p -h ubuntu
   slave stop;
   CHANGE MASTER TO MASTER_HOST = '10.161.159.166', MASTER_USER = 'replicator',
   MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin.000006', MASTER_LOG_
   POS = 12927236;
   slave start;
   ```

   On geotest2:

   ```
   mysql -u root -p -h centos
   slave stop;
   CHANGE MASTER TO MASTER_HOST = '10.161.159.189', MASTER_USER = 'replicator',
   MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin.000002', MASTER_LOG_
   POS = 85527658;
   slave start;
   ```

The MySQL databases are now connected and replicated. You can confirm the replication by sending an MT message to geotest1, then confirm that geotest2 received the delivery report. Have geotest1 px21 SMS plug-in binding transmitter only, and geotest2 px21 SMS plug-in binding receiver only.

GeoStorageService Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services** followed by **GeoStorageService**. Alternately, use a Java application. For information on the methods and fields of the supported MBeans, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

GeoRedundantService Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services**, and then **GeoRedundantService**. Alternately, use a Java application. For information on the methods and fields of the supported MBeans, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

7

Configuring Coherence Clusters

This chapter describes how to configure the Oracle Communications Services Gatekeeper Oracle Coherence storage provider.

Understanding Coherence

Coherence is used for clustering in Services Gatekeeper. The storage service uses Coherence for cluster-wide storage.

In a basic domain configuration, a default Coherence configuration is created when you create the domain. If you support a clustered environment, you must create a Coherence configuration.

Coherence supports two different types of addressing:

- Multicast, where all messages are broadcast to all servers in the network.
- Unicast, where the servers in the network tier are aware of the IP-addresses, or host-names, of all servers in the tier and only send messages to this well-known set of addresses.

Configure the Coherence cluster-related attributes such as multicast address and port, unicast address, and port overrides using the administration console. Oracle WebLogic Server stores the attributes in the WebLogic configuration repository and makes them available to the Coherence cluster service.

Configuring Coherence

To configure Coherence:

1. Log in to the administration console.
2. Expand the **Services** section in the **Domain Structure** pane.
3. Click **Coherence Clusters**.
4. Click **New**.

   The **Create Coherence Cluster Configuration** screen is displayed.
5. Enter `Coherence-OCSG` in the **Name** field. This is a fixed value and must be entered exactly as stated.
6. Ensure that **Use a custom Cluster Configuration File** is not selected.
7. Click **Next**.

   The **Coherence Cluster Addressing** page is displayed.

8. For unicast addressing:

   a. In the **Unicast Listen Address** field, enter the IP address for the cluster unicast listener.

   b. In the **Unicast Listen Port** field, enter the port for the cluster unicast listener.

   c. If the port cannot be bound because it is already in use, configure the unicast port number to be automatically incremented.

      To do so, select the **Unicast Port Auto Adjust** check box.

9. For multicast addressing:

   a. In the **Multicast Listen Address** field, enter the IP address for the cluster multicast listener.

   b. In the **Multicast Listen Port** field, enter the port for the cluster multicast listener.

10. Click **Next**.

    The **Coherence Cluster Targets** page is displayed.

11. Select all network tier servers as Coherence Cluster targets by selecting the check-box in front of *Network_Tier_Cluster_Name*, where *Network_Tier_Cluster_Name* is the name of your network tier cluster.

12. Click the **All servers in the cluster** radio-button in the *Network_Tier_Cluster_Name* group.

13. Click **Finish**.

    The **Summary of Coherence Clusters** page is displayed.

14. Click **Coherence-OCSG**.

    The **Settings for Coherence-OCSG** page is displayed.

15. Click the **Well Known Addresses** tab.

    The **Well Known Addresses** page is displayed.

16. For each network tier server:

    a. Click **New**.

       The **Coherence Cluster configuration Properties** page is displayed.

    b. Enter the name of the Coherence cluster configuration in the **Name** field.

    c. Enter the IP address to listen to in the **Listen Address** field.

    d. Enter the port to listen to in the **Listen Port** field.

    e. Click **OK**.

8

Managing Users and User Groups

This chapter describes how to set up and manage the Oracle Communications Services Gatekeeper administrative users.

About Services Gatekeeper Users and User Groups

Services Gatekeeper classifies its users as either Traffic users or Management users.

- Traffic users are users (application instances) who use the application-facing interfaces to send traffic through Services Gatekeeper. Traffic users cannot login to the Administration Console or perform any management operations.
- Management users are users who have access to and can perform management and administration functions. Management users are identified by their *user type*. Each management user is also assigned a *user level*.

The PRM Portals create these users when you create users, groups, apis, applications, and network service suppliers. Oracle recommends that you use either the PRM portals to create users for a Services Gatekeeper implementation, or the MBeans listed in this chapter but not both to avoid unintentionally invalidating users accounts.

The Services Gatekeeper installation process creates default user groups in the WebLogic Server Embedded LDAP server. Table 8–1 lists the names of the default user groups, their membership criteria, and classification of user roles.

Table 8–1 User Groups and Privileges

User Group Name	Membership and Privileges	Role
Traffic User	All application instances belong to this group. ■ They should be able to just send traffic and should not have access to management functions. ■ They should not have access to WebLogic Server or Services Gatekeeper MBeans. ■ They should not be able to log into the console and perform WebLogic Server administration operations.	TrafficUser
OamUser	Management users who are of OAM type ■ They have access to the console based on their level. ■ They should not be able to send traffic.	OamUser
PrmUser	Management users who are of PRM type ■ They should not have access to the console. ■ They should perform their management operations using the PRM interfaces.	PrmUser

About User and Group Roles in the Production Environment

Each group contains a user or set of users and is associated with a security role. Groups are generally static; they do not change at run time. A basic role condition can include users or user groups in a particular security role. For example: *set Admin Role to all users in Administrators group*. A policy contains one or more conditions. For example, a simple policy can be *Allow access if the user belongs to Admin Role*.

Roles are evaluated at run time by the Role Mapping Provider by checking the authenticated subject.

In a Services Gatekeeper production environment, Services Gatekeeper handles traffic from application instances (traffic users). When an application instance sends a Simple Object Access Protocol (SOAP) request to the application-facing interfaces, the WebLogic network gatekeeper (WLNG) Application Authenticator authenticates the application instance. Upon successful authentication, the WLNG Application Authenticator adds the Traffic User group, the service provider ID, application ID, service provider group ID, and application group ID to the user principal (identity in the realm). That identity determines the access rights of the application instance in the system.

When management users log in successfully to a Services Gatekeeper Administration Console, they are added to the Oam User group with predefined access rights to the system.

User Types

Following are the predefined management user types:

- **Administrative users** use the Administration Console or Java Management Extensions (JMX) to interact with Services Gatekeeper.
- **PRM operator users** use the Partner Relationship Management (PRM) Operator web services interfaces to interact with Services Gatekeeper.
- **PRM service provider users** use the PRM Service Provider web services interfaces to interact with Services Gatekeeper.
- **PRM Network Service Supplier** users use the PRM Network Service Supplier Portal to create interfaces.

When creating a management user, the user is mapped to the Weblogic Server authentication provider WLNG Operation, Administration, and Maintenance (OAM) Authenticator.

About User Levels

Management users are assigned different user levels based on which JMX resources they will be able to access. Table 8–2 lists the access privileges associated with user levels on Services Gatekeeper and WebLogic Server.

Table 8–2 User Levels and Privileges

User Level	Access on Services Gatekeeper	Access on WebLogic Server
1000	Administration access to management functions	Administration access: ■ View, modify, and administer server configuration. ■ Deploy applications. ■ Start, resume and stop servers.
666	read/write access on management functions	Deployer access: ■ View the server configuration, including some encrypted attributes related to deployment activities. ■ Change startup and shutdown classes, Web applications, JDBC data pool connections, EJB, Java EE Connector, web service. If applicable, edit deployment descriptors. ■ Access deployment operations in the Java EE Deployment Implementation (JSR-88).
333	Read-only access on management functions	Monitor access: ■ View the server configuration. ■ Have read-only access to Administration Console, WLST, and other MBean APIs.
0	No access to management functions; Assigned to PRM Service Provider users internally.	Anonymous access: No access to the console

About User Management Methods

As a system administrator, you belong to a group of administrative users who manage Services Gatekeeper and its users. Table 8–3 provides an overview of the operations that administrative users employ to oversee the users of their Services Gatekeeper installation.

Table 8–3 Operations Associated with Management Tasks

To...	Use this Method in ManagementUserMBean
Create an administrative user	**addUser**
Change password	**changeUserPassword**
Delete an administrative user	**deleteUser**
Get user level	**getUserDescription**
List administrative users	**listUsers**

For details of these methods, see in the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Managing Access Privileges Through Policies

As an administrator you can restrict access to a subset of management interfaces by applying eXtensible Access Control Markup Language (XACML) policies.

To apply these policies to add more granular access control:

1. Add a new management user.
2. Create a user group.
3. Add the user to the user group.
4. Add an XACML policy to assign a role to the group.
5. Add an XACML policy to the user group. Restrict access at the desired level such as MBean, MBean attribute, or MBean operation level. See “Understanding WebLogic Resource Security” in *Oracle WebLogic Server Securing Resources using Roles and Policies for Oracle WebLogic Server* for a detailed description of this process.

   The basic process includes:

   – Determine a special identifier, the *resourceId,* for each MBean.

   – Create an XACML policy for the new security role.

   – Specify one or more rule elements that define which users, groups, or roles belong to the new security role.

   – Attach this role to the MBean using the *resourceId*.

You access the **ManagementUserMBean** and **ManagementUserGroupMBean** MBeans from the Administration Console (**OCSG > AdminServer > Container Services > ManagementUsers**).

For more information, see the entries for **ManagementUserMBean** and **ManagementUserGroupMBean**, in the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

9

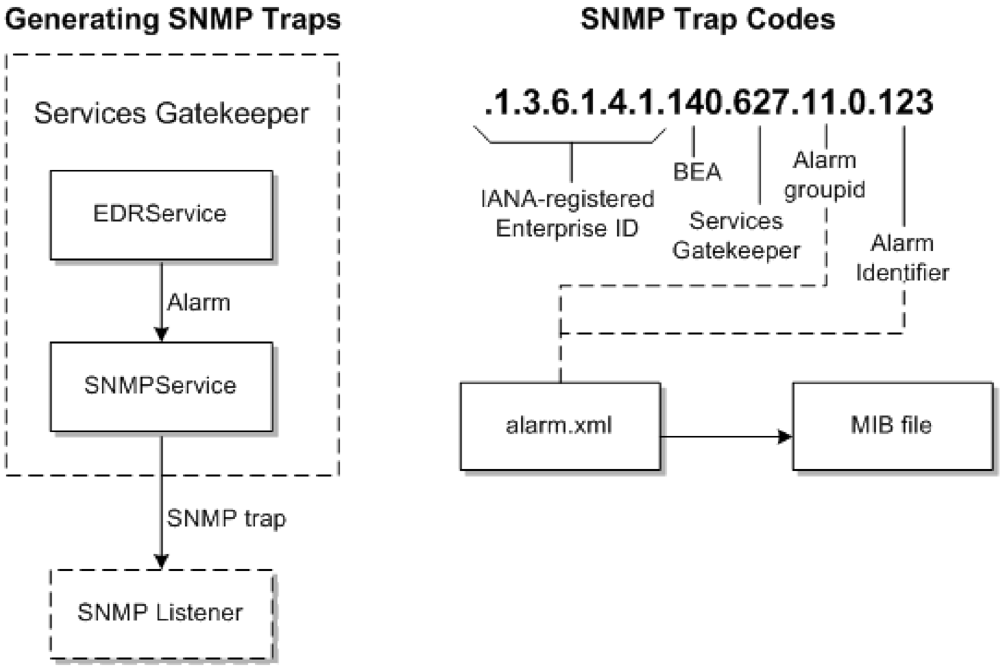
Configuring and Using the SNMP Service

This chapter describes how to configure and use the Simple Network Management Protocol (SNMP) service in Oracle Communications Services Gatekeeper.

Understanding the SNMP Service

The SNMP service collects alarms and distributes them as SNMP traps. Figure 9–1 illustrates SNMP trap generation and explains the trap codes.

Figure 9–1 SNMP Overview



The SNMP service acts as an internal alarm listener and sends traps (or notifications) to any registered SNMP trap listener. There is a one-to-one relationship between alarms and SNMP traps. The Management Information Base (MIB) file defining the SNMP traps is based on the content of the **alarm.xml** file. See "Generating SNMP MIB Files" for instructions on how to create these files.

Each individual alarm ID generates the object identifier for each SNMP trap.

The SNMP traps sent consist of:

IANA-registered private enterprise ID + BEA ID + Services Gatekeeper ID + "0" + alarm identifier

- The IANA-registered enterprise ID is configurable using the **EnterpriseObjectIdentifier** field in **SNMPServiceMBean**.
- The BEA ID is static, and the value is 140.
- The Services Gatekeeper ID is static, and the value is 627.
- The alarm identifier is defined in **alarm.xml**, in the attribute `id` of the alarm element.

The default **BEA-WLNG-MIB** MIB file is located in *Gatekeeper_home*/**ocsg_pds**_*release_level*/**integration**, where *Gatekeeper_home* is the directory in which Services Gatekeeper is installed. You must create a new MIB file each time you change.

Configuring and Managing the SNMP Service

This section describes the following tasks required to configure and manage the SNMP service:

- Configuring SNMP Service
- Generating SNMP MIB Files

See "SNMPServiceMBean Reference" for information on using the **SNMPServiceMBean.**

Configuring SNMP Service

Table 9–1 lists the attributes used to configure the SNMP Service.

Table 9–1 Task Overview

To define	Use This Field in SNMPServiceMBean
The SNMP Community string.	**Community**
Enterprise Object Identifier. This attribute has a default value and should normally not be changed.	**EnterpriseObjectIdentifier**
SNMP version to use.	**SNMPVersion**
Which alarm severity levels that shall be distributed as SNMP traps.	**SeverityFilter**

Define trap receivers using the **addTrapReceiver** method in **SNMPServiceMBean**.

Trap Receivers

Table 9–2 lists the operations on trap receivers.

Table 9–2 Task Overview

To	Use These Methods in SNMPServiceMBean
Add a trap receiver	**addTrapReceiver**
List defined trap receivers	**listTrapReceivers**

Table 9–2 (Cont.) Task Overview

To	Use These Methods in SNMPServiceMBean
Delete an existing trap receiver	First, call **listTrapReceivers** to list the registered trap receivers. Locate the id of the trap receiver to delete. Then, call **deleteTrapReceiver** with the ID of the trap receiver to delete as an input parameter.

See "SNMPServiceMBean Reference" for information on using the **SNMPServiceMBean**.

Generating SNMP MIB Files

The MIB file specifies the alarms that you forward as SNMP traps. When you change the alarm descriptor, you must generate a new MIB file and distribute it to your SNMP clients. You do this by copying the contents of the alarm descriptor and pasting it into an xml file. Use this xml file to generate the MIB file.

You generate the management interface base (MIB) file corresponding to the alarms using the **mibgenerator** Apache Ant task defined in **com.bea.wlcp.wlng.ant.MIBGeneratorTask**.

Unless stated otherwise, perform these steps in the *Middleware_home*/**ocsg_pd***release_level*s/**integration** directory. Where *Middleware_home* is the directory that serves as the repository for common files that are used by Oracle Communications products installed on the same machine, such as Services Gatekeeper and WebLogic Server.

To generate a new MIB file:

1. Create a **/tmp** directory.
2. Copy the example **alarm_example/edr-config.xsd** file to the **/tmp** directory.
3. Open the *Gatekeeper_home*/**user_projects/domains/basic-domain/config/custom/wlng-edr.xml** file for editing.
4. Save the **alarm-config** portion of the *Gatekeeper_home*/**user_projects/domains/basic-domain/config/custom/wlng-edr.xml** file to a file called **alarm.xml**. in the **/tmp** directory.
5. Record the **edrDir** entry in *Middleware_Home*/**ocsg_pds**_*release_level*/**integration/build.xml** so that you can recreate it.
6. Change the **edrDir** directory in the example build file, *Middleware_Home*/**ocsg_pds**_*release_level*/**integration/build.xml** to reference the **/tmp** directory using this example:

   edrDir=/tmp
7. Navigate to *Middleware_Home*/**ocsg_pds**_*release_level*/**integration/** and run the

   ant generate-mib

   command to generate a new **BEA-WLNG-MIB_Generated** file containing the new list of alarms.
8. Return the **edrDir** item in *Middleware_Home*/**ocsg_pds**_*release_level*/**integration/build.xml** to it's original setting.
9. Distribute the new **BEA-WLNG-MIB** file to all of your SNMP clients.

SNMPServiceMBean Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services**, and then **SNMPService**. Alternately, use a Java application. For information on the methods and fields, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

10

Generating Statistics for Transaction Licenses

This chapter describes how to generate and manage transaction statistics that Oracle Communications Services Gatekeeper uses for licensing purposes.

Services Gatekeeper also provides a variety of API and application usage reports through the PRM Portals. For more information and instructions on how to implement them, see “Managing Application and API Usage with Report Statistics” in *Services Gatekeeper API Management Guide*.

About Generating Statistics and Reports

Services Gatekeeper keeps usage statistics on the number of transactions handled over time. Transactions are grouped into transaction *types*. Transaction types are used for calculating usage costs and grouping reports. Transaction types are in turn grouped into different categories. For more information on transaction types, see "Transaction Types" and *Services Gatekeeper Licensing Guide*.

These statistics are generated by communication services. Verification mechanisms ensure that all network protocol plug-ins have the statistics aspects applied. This verification takes place when the plug-in is deployed into Services Gatekeeper.

Statistics are held in an in-memory store and are periodically flushed to database. Statistics reports are created based on information in the database. You can get a snapshot of the current status of the transaction, or request, counters from the in-memory store.

Understanding Statistics Reports

Statistics are used to generate reports filtered on a number of parameters:

- Time interval
  - Start time
  - End time
- Individual statistics types or an aggregate of all statistics types.
- Originator of the transaction:
  - Service provider account ID
  - Application account ID
- Per cluster or per server

Note: Not all combinations of the above are supported.

You can retrieve reports in the following ways:

- View the report in the administration console. See "Accessing the System Report from the Console".
- Access online. See "Retrieving the System Report as a File".
- Obtain a system report. See "Retrieving the Weekly System Report".
- Obtain a transaction log report. See "Retrieving the Transaction Usage Log Report".

Accessing the System Report from the Console

This report is created by the **getSystemStatistics** method in **StatisticsServiceMBean**. The output is presented in the console.

See "StatisticsServiceMBean Reference" for information on using **StatisticsServiceMBean**.

Retrieving the System Report as a File

This report is created by the **saveStatisticsToFile** method in **StatisticsServiceMBean**. The format is adapted for programmatic processing of the file.

Retrieving the Weekly System Report

The weekly report is a predefined report. It shows the total number of transactions through Services Gatekeeper hour by hour during a specified week. It also shows total usage for each day and the average transaction rate (transactions/second) during the busy hour of each day. The busy hour is defined as the 60 minutes during which the largest number of transactions are handled, and it does not depend on clock hours. Any 60-minute period (5-minute intervals are used) can be identified as the busy hour.

A weekly system statistics report shows:

- The total number of transactions during the specified week
- The number of transactions during each hour of the days in the week
- The number of transactions during each day of the week
- The transaction rate (transactions/second) during the busy hour of each day

This report is stored on file.

Retrieving the Transaction Usage Log Report

A transaction usage log (called the **license_limit** log) can be extracted from Services Gatekeeper, which records the transactions on which usage costs are based. The log file contains a set of entries, where each entry represents the average transactions per second during the busy hour of a 24-hour period starting at 12:00 AM and ending 11:59 PM the previous day.

The transaction usage log report is an XML file with a header and a footer.

Example 10–1 Structure of license_limit log file

```
<transaction_limit_log>
<start> </start>
<end> </end>
<log_entry>
</log_entry>
<checksum>
</checksum>
</transaction_limit_log>
```

All information is contained within the `<transaction_limit_log> element`.

A header, encapsulated by the tag `<header>`, contains information on the time period over which the log is generated, with a start and end date in the tags `<start>` and `<end>`, respectively. The format is DD-MM-YYYY.

Directly following the tag `</end>`, one or more log entries are found in the tag `<log_entry>`.

There is one `<log_entry> element` created for each day and transaction type.

This element contains a set of attributes:

- **group**: The transaction group for which it is valid. Possible values are Platform or Oracle-modules.
- **start**: The start date and time for the busy hour. Format is YYYY-MM-DD HH:MM, where HH is in 24-hour format.
- **end**: The end date and time for the busy hour. Format id YYYY-MM-DD HH:MM, where HH is in 24-hour format.
- **tps**: The average transactions per second during the busy hour. This value is compared with contractual levels by the file auditor to determine usage costs.
- **limit**: No longer used. Filled with dummy value.
- **exceeded**: No longer used. Always false.

A checksum is contained in the `<checksum>` element. The checksum is created based on the content of the file. The checksum is used for validating that the file has not been changed.

Example 10–2 License limit log file example

```
<transaction_limit_log>
<start>2006-01-01</start>
<end>2006-01-31</end>
<log_entry group="BEA-modules" start="2007-01-01 13:45" end="2007-01-01 14:45"
tps="27.35" limit="50" exceeded="false"/>
<checksum>f8b904410896b3f92159524c6c68</checksum>
</transaction_limit_log>
```

For more information about licensing, see *Services Gatekeeper Licensing Guide*.

Understanding Counter Snapshots

Counter snapshots are real-time snapshot of the statistics counters. To see the counter snapshot for the local server, use the **CounterSnapshot** (r) field to **StatisticsServerMBean**.

Table 10–1 describes the different categories into which the snapshot is organized.

Table 10–1 Snapshot Categories

Counter snapshot category	Description
Per server	Sum of the statistics counters for all requests regardless of the originator of the request
Per server and service provider	Sum of the statistics counters for all requests originating from a given service provider, regardless of application
Per server and service provider and application	Sum of the statistics counters for all requests originating from a given service provider and application

The output is in the form of key/value pairs, described below in Table 10–2.

Table 10–2 Snapshot Key-Value Pairs

Key	Value
Source	Server name
applicationIdentifier	Application ID; Empty if counter snapshot category is per server or per server and service provider
serviceProviderIdentifier	Service provider ID; Empty if counter snapshot category is per server or per server and service provider
type	Statistics type
num of transactions	The snapshot of the counter

See "StatisticsServiceMBean Reference" for information on using **StatisticsServiceMBean**.

Managing Statistics

The following sections describe how to manage statistics **StatisticsServiceMBean**.

Configuring the Statistics Time Interval

Table 10–3 describes how to configure statistics persistence interval.

Table 10–3 Configuring Statistics Persistence Interval in StatisticsServiceMBean

To configure...	Use
Statistics persistence interval	The **StoreInterval** field.

Configuring Statistics Types and Transaction Types

Table 10–4 describes the operations to configure statistics types.

Table 10–4 Operations to Configure Statistics types in StatisticsTypeMBeans

To configure...	Use
Add a new type	The **addStatisticType** method
Remove an existing type	The **removeStatisticType** method

Table 10–4 (Cont.) Operations to Configure Statistics types in StatisticsTypeMBeans

To configure...	Use
List existing	The **removeStatisticType** method The **listStatisticTypes** method The **listStatisticTypeDescriptors** method

Viewing In-Flight Statistics counters

Table 10–5 shows the attribute used to configure a subset of the statistics counter.

Table 10–5 Attribute to Configure Subset of Statistics Counter in StatisticsServiceMBean

To Configure...	Use
A subset of the statistics counters	The **CounterSnapshot** (r) field

Generating Statistics Reports

Table 10–6 shows the operations used to generate statistics reports.

Table 10–6 Operations to Create Statistics Reports Using StatisticsServiceMBean

To generate a...	Use
Transaction usage log report	The **createLicenseLimitLog** method.
Weekly report	The **createWeeklyReport** method
Statistics summary over a time interval	The **getStatistics** method
Statistics summary over the last minutes	The **getSystemStatistics** method
Statistics report to file	The **saveStatisticsToFile** method The **saveAccountStatisticsToFile** method

See "StatisticsServiceMBean Reference" for information on using **StatisticsServiceMBean**.

Add Usage Thresholds

Table 10–7 shows the attributes used to add usage thresholds.

Table 10–7 Attributes to Add Usage Thresholds in StatisticsServiceMBean

To add a...	Use
Limit alarm threshold for Oracle module-based TUPS	The **ModuleBHTUPSThreshold** field
Limit alarm threshold for platform-based TUPS	The **PlatformBHTUPSThreshold** field

See "StatisticsServiceMBean Reference" for information on using **StatisticsServiceMBean**.

Transaction Types

A set of transaction types are defined for the each of the communication services that come as a part of Services Gatekeeper. The transaction type **TRANSACTION_TYPE_EXTENSION** is used for new communication services, developed as extensions to Services Gatekeeper.

For a description of the events that generate statistics for a certain transaction type, see the discussion about the statistics for the communication service in *Services Gatekeeper Communication Service Reference Guide*.

The information includes correlation maps between methods invoked from either an application or the telecommunication network and the corresponding transaction type. See *Services Gatekeeper Communication Service Reference Guide* for more information.

StatisticsServiceMBean Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services**, and then **StatisticsService**. Alternately, use a Java application. For information on the methods and fields of the supported MBeans, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

11

Managing and Configuring EDRs, CDRs and Alarms

This chapter describes how to manage and configure Oracle Communications Services Gatekeeper Event Data Records (EDRs), Charging Data Records (CDRs), and alarms.

About EDRs, CDRs, and Alarms

You use Event Data Records (EDRs) and Charging Data Records (CDRs) to capture specific events that occur while traffic is running between applications, Services Gatekeeper, and the resource or network node the application is using. You can use EDRs and CDRs to charge your users for specific events that they consume. The consumption can be volume-based, time-based, event-based, or any other combination of EDR fields that you choose.

EDRs are the base component of both CDRs and alarms.

EDRs are generated:

- Automatically, using aspects at various locations in a network protocol plug-in.
- Manually, anywhere in the code using **EDRServiceMBean** directly.

Alarms warn you of traffic processing problems.

See *Alarms Handling Guide* for details on the individual alarms organized by **tagalarm** number.

Understanding EDRs

All EDRs are passed through the **EDRServiceMBean**. All EDRs are dispatched to a JMS- distributed topic so external clients can receive them over JMS. EDRs are *not* persisted in the database.

See "EDRServiceMBean Reference" for information on using the **EdrServiceMBean**.

Understanding Alarms

Alarms are EDRs that are mapped to alarms using the **alarm.xml** configuration file: see "EDR Categories and XML Markup".

You can configure Alarms to persist. See "**StoreAlarms**" in **EdrServiceMBean** in *OAM Java API Reference* for details.

The *Alarms Handling Guide* for lists all alarms by **tagalarm** number, and contains information on how to solve invidivual problems.

Understanding CDRs

CDRs are EDRs that are mapped to CDRs using the **wlng-edr.xml** configuration file. See "EDR Categories and XML Markup" for details.

You can configure CDRs to persist. See "**StoreCDRs**" in **EdrServiceMBean** in *OAM Java API Reference* for details.

See "EDRServiceMBean Reference" for information on using the **EdrServiceMBean**.

Understanding External EDR listeners

External EDR listeners are JMS topic subscribers. See the discussion on "Creating EDR Listeners" and *Services Gatekeeper Extension Developer's Guide* and "Generating SNMP MIB Files" for more information.

EDR Categories and XML Markup

In order to categorize the objects in the EDR flow as either pure EDRs, alarms or CDRs, the EDR service uses an EDR configuration file:

domain_home/**config/custom/wlng-edr.xml**

In the configuration file:

- The `<edr-config>` element contains descriptors that describe pure EDRs.
- The `<alarm-config>` element contains descriptors that describe EDRs that are considered to be alarms.
- The `<cdr-config>` element contains descriptors that describe EDRs that are considered to be CDRs.

The default Services Gatekeeper installation comes with a set of predefined descriptors. Change and adapt the descriptors as part of an integration project.

You can edit and reload the XML configuration file using the EDR Configuration pane. See "Managing EDR/CDR/Alarm Configuration Files" for details.

Example 11–1 illustrates the structure of **wlng-edr.xml**. See the discussion on "Creating EDR Listeners" and *Services Gatekeeper Extension Developer's Guide* and "Generating SNMP MIB Files" for more information.

Example 11–1 structure of wlng-edr.xml

```
<edr-config xsi:schemaLocation="http://www.bea.com/ns/wlng/30 edr-config.xsd">
<edr id="ID" description="description">
<filter>
<method>
<name>message response message</name>
<class>fully qualified class name</class>
</method>
</filter>
</edr>
.....
</edr-config>
```

Understanding the EDR format

The following values are always available in an EDR when it is generated in the standard way:

- Class name
- Method name
- Direction (application-facing, network-facing), if the request is travelling from Services Gatekeeper to the network or from the network to Services Gatekeeper
- Position (before, after), if the EDR was emitted before or after the method was invoked or the exception was thrown
- Interface (application-facing or network-facing), if the EDR was emitted from the application-facing interface or from the network-facing interface of the plug-in
- Source (method, exception), if the EDR is related to a method invocation or to an exception

In addition to these values, the EDR may also contain values relevant to the context of the request.

Figure 11–1 Application Tier and Network Tier EDR Generation

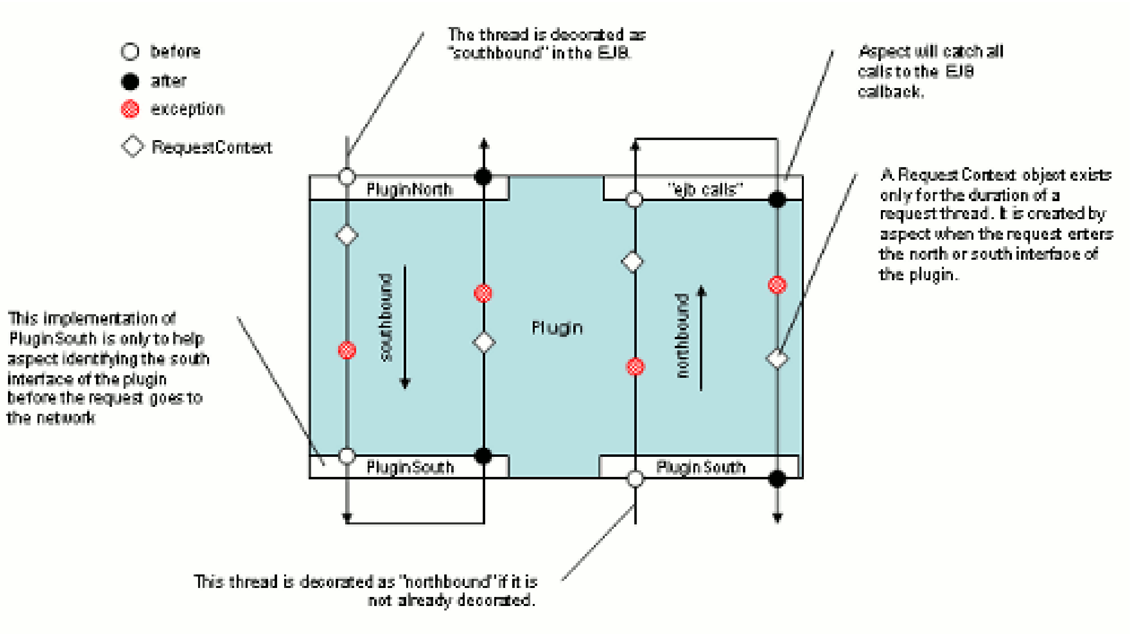

○ before
● after
exception
◇ RequestContext
The thread is decorated as "southbound" in the EJB.
Aspect will catch all calls to the EJB callback.
A Request Context object exists only for the duration of a request thread. It is created by aspect when the request enters the north or south interface of the plugin.
PluginNorth
"ejb calls"
southbound
Plugin
northbound
This implementation of PluginSouth is only to help aspect identifying the south interface of the plugin before the request goes to the network
PluginSouth
Plugin South
This thread is decorated as "northbound" if it is not already decorated.

Table 11–1 describes the contents of an EDR. Individual value fields in an EDR are retrieved by name using a key in a name/value pair.

Table 11–1 Contents of an EDR

String value of name (key) in name/ value pair	Description
EdrId	Defined in **wlng-edr.xml**
ServiceName	The name or type of the service
ServerName	Name of server where the EDR was generated
Timestamp	The time at which the EDR was triggered Milliseconds since midnight, January 1, 1970 UTC

Table 11–1 (Cont.) Contents of an EDR

String value of name (key) in name/ value pair	Description
ContainerTransaction Id	WebLogic Server transaction ID (if available)
Class	Name of the class that logged the EDR
Method	Name of the method that logged the EDR
Direction	Direction of the request
Source	The type of source that logged the EDR
Position	Position of the EDR relative to the method that logged the EDR
State	Where the EDR was dispatched
Interface	Interface where the EDR is logged
Exception	Name of the exception that triggered the EDR
SessionId	Session ID
ServiceProviderId	Service provider account ID
ApplicationId	Application account ID
AppInstanceId	Application account identifier
TransactionId	Transaction Id. Correlates completed traffic among all three EDR states
Facade	Facade (REST or SOAP)
OrigAddress	The originating address with scheme included. For example: tel:1212771234
DestAddress	The destination address, or addresses, with scheme included
<custom>	Any additional context-specific information
PluginID	The unique ID of the plug-in instance
URL	The URL of the current web service
WebAppName	Name of the current web application
HttpMethod	HTTP method, such as "POST", or "GET"
SubscriberID	Subscriber (user) identifier (using route address)

Using EDRtoAnalytics

EdrToAnalytics provides you with a statistical picture of the number of EDRs between the EDR processor and the data store writer.

You have these options for using these statistics:

- Connect the WebLogic JMS Monitor to a Services Gatekeeper NT server and view these statistics. You can inspect the total messages published by **EdrService**. The number of current message indicates how many message are buffered by JMS service before they are delivered to users. If this number gets very large or continuously increases then you know you have a problem.
- View the **EdrService** statistics from the Services Gatekeeper Administration Console. This metric shows you the total number of EDRs published by

EdrService and a snapshot of the number of EDRs being queued between the EDR publisher and the JMS server.

- View the **EdrToAnalytics** statistics in the Services Gatekeeper Administration console. This metric shows you the total accumulation of EDR received by each **EdrListener**, and the snapshot of the usage of queue between **EdrProcessor** and data store writer.

Reference: Attributes and Operations for EdrToAnalytics

Managed object: **EdrToAnalytics**

MBean: **oracle.ocsg.edr.analytic.management.EdrAnalyticMBean**

Operation: listEdrListnerStatistics

Scope: Cluster

Retrieves the number of EDR messages received and written to the database by the **EdrToAnalytics** MBean.

Signature:

```
listEdrListenerStatistics()
```

Using EDRService

The following sections explain how to use **EDRService**.

Configuring EDRService

To configure the behavior of the EDRService, in the **EdrServiceMBean**:

1. Specify **PublishToJMS**"
2. Specify **StoreAlarms**.
3. Specify **StoreCDRs**.

See "EDRServiceMBean Reference" for information on using the **EdrServiceMBean**.

Managing EDRService

The following section describes how to manage the EDRService.

Defining batch attributes

To configure the maximum number of EDRs sent in a batch to a JMS EDR listener and the maximum time to wait before the EDRs in the buffer are sent to listeners:

1. Specify **BatchTimeout**.
2. Specify **BatchSize**.

See "EDRServiceMBean Reference" for information on using the **EdrServiceMBean**.

EDRServiceMBean Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services** followed by **EdrService**. Alternately, use a Java application. For information on the methods and fields of the supported MBeans, see the “All Classes” section of *Services Gatekeeper OAM Java API Reference*.

Managing EDR/CDR/Alarm Configuration Files

Administrators use the Services Gatekeeper **EDR Configuration** pane of the Services Gatekeeper Administration Console to load new EDR, CDR, and alarm configuration files.

Open the pane the Domain Structure in the Administration Console. Select **OCSG**, then *Server Name*, and then **EDR Configuration**.

Note: You must click Lock and Edit first.

Table 11–2 describes the inputs for the entry fields in the configuration pane.

Table 11–2 Services Gatekeeper EDR Configuration pane

Entry field	Input
EDR descriptor	The EDR configuration file. See "EDR Categories and XML Markup".
CDR descriptor	The CDR configuration file. See "EDR Categories and XML Markup".
Alarm descriptor	The alarms configuration file. See "EDR Categories and XML Markup".

12

Resolving Policy Deny Codes

This chapter explains the policy deny codes generated by Service Interceptors in Oracle Communications Services Gatekeeper.

Understanding the Policy Deny Codes

Policy deny codes are embedded in policy exceptions thrown to applications, embedded in CDRs and alarms.

Policy exceptions are thrown when a usage policy has been violated by an application, for example if a parameter provided by an application is out of bounds or if the request rate has been exceeded.

In policy exceptions thrown toward applications, the deny code is embedded in the exception.

In alarms, the policy deny code is present if the alarm originated from a policy violation.

In CDRs, the policy deny code is present in the additional information field, enclosed by `<denyCode>` elements. For example: `<denyCode>21</denyCode>`.

Policy Deny Code Values

Table 12–1 contains a list of policy deny codes data generated when a policy deny occurs.

Table 12–1 Policy deny data

Policy Deny Code	ID of Corresponding Alarm	TagPolicy Field in Alarm	Description
-1	Not applicable	Not applicable	Unspecified
0	Not applicable	Not applicable	Unspecified
1	102826	1001	Application instance does not exist
2	102826	1001	Application instance is not active
3	102826	1001	Application does not exist
4	102826	1001	Service provider account does not exist

Table 12–1 (Cont.) Policy deny data

Policy Deny Code	ID of Corresponding Alarm	TagPolicy Field in Alarm	Description
6	102827	1002	Unable to get service provider and application information
7	102823/3026	2003	Application request limit exceeded
8	102823/3026	2003	Service provider request limit exceeded
9	102828	2001	Application request limit exceeded for Service Type
10	102828	2001	Service provider request limit exceeded for Service Type
11	102826	1001	Service provider account not active.
12	102826	1001	Application instance not active
13	102822/3025	2002	Service provider quota limit exceeded
14	102822/3025	2002	Application quota limit exceeded
15	102829	2008	Service provider quota limit exceeded for Service Type
16	102829	2008	Application quota limit exceeded for Service Type
20	102830	4001	All properties denied
21	102831	3001	Parameter value is not allowed
22	102832	4002	Request info empty
23	102833	3002	Accessing the method not allowed
24	102834	3003	No service provider group SLA found for application instance
25	102834	3003	No application group SLA found for application instance
26	102835	4003	Exception thrown calling correlator
27	102836	4004	Exception thrown calling factory
28	102824/3027	2004	Global node or Service provider node request limit exceeded
29	102825/3028	2005	Global or service provider node contract is out of date
30	102837	3004	No global or service provider node SLA found

Table 12–1 (Cont.) Policy deny data

Policy Deny Code	ID of Corresponding Alarm	TagPolicy Field in Alarm	Description
31	102838	3005	Application or Service provider group service contract is out of date
32	102839	3006	Application or service provider Service Type contract is out of date
33	102834	3003	No SLA found
34	102840	3007	No Service Contract found
35	102841	2006	User restricts all
36	102842	2007	User quota limit reached
37	102842	2007	User rate limit reached
38	102824	2004	Global node request limit exceeded
41	102824	3010	Application request quota limit exceeded
44	102824	3010	Application request rate limit exceeded
104	Not applicable	Not applicable	The request has been denied through the credit control interceptor

13

Managing and Configuring the Trace Service

This chapter describes how to configure and manage the Trace service in Oracle Communications Services Gatekeeper.

Understanding the Trace Service

The Trace service is based on Log4J. Services Gatekeeper maintains a log file, **default.log**, and each service instance writes to this log file on its local file system. The trace service writes log files in the *domain_home*/**servers**/*server name*/**trace** directory.

Understanding Basic Tracing

For basic tracing, the root logger `rootdefault` maintains the trace file.

Use these fields and methods to the **TraceServiceMBean** to configure and maintain basic tracing:

- **TracingEnabled** field
- **attachAppender** method
- **flushBuffers** method
- **rollOver** method

See "TraceServiceMBean Reference" for information on using this MBean.

Understanding Context Tracing

Context tracing generates log messages filtered on the context of a request such as, for a certain service provider or application, in addition to basic tracing. Context trace filters narrow the generated log messages to requests that match a given context filter. A context filter has predefined filter types that define what to filter on. For a given filter type, the value to match is defined. This is used for tracing on individual service providers, applications, and so on.

New context trace files can be added, and context filters and context categories can be applied to these files. Context categories can be added to the context trace file to log messages from one or more Services Gatekeeper services. For example, to log messages from the Budget service and the Short Message Peer-to-Peer Protocol (SMPP) plug-in to a context trace file, the context categories for these services are added to the context trace file.

To define a context trace file:

1. In the Services Gatekeeper Administration Console, choose **Container Services** then **TraceService**.

2. Select **createContextTraceFile**, or **createRootContextTraceFile**.
3. For each context category to add for the context trace file, use the **addContextCategory** method to **TraceServiceMBean**
4. For each context filter to add, use the **addContextFilter** method

See "TraceServiceMBean Reference" for inforamtion on using this MBean.

Log4J Hierarchies, Loggers, and Appenders

There is a set of Log4J Dynamic MBeans shipped with Services Gatekeeper that come by default with appenders. Appenders are responsible for delivering LogEvents to their destination.

One Log4J hierarchy and one location are defined and displayed in the Services Gatekeeper Administration Console under **Log4J** as:

- log4j:hiearchy=default,Location=AdminServer

The rootDefault logger is connected, to the default hierarchy:

- log4j:Location=AdminServer,logger=rootDefault

You can add loggers, and add appenders to the loggers. You can change both the priority of the logger and the appender to use. You can also configure parameters for the loggers and appenders.

The set of default appenders are defined as follows:

- log4j:appender=default,Location=AdminServer
- log4j:appender=default,Location=AdminServer

Note: The Log4J attributes are dynamic and not persisted.

To persist Log4J settings, use the configuration file:

$Domain_Home/**log4j/log4jconfig.xml**

Configuring Trace for Access Tier servers

Trace configuration for network tier servers is performed through the Services Gatekeeper Administration Console by accessing **Log4J** for a server. For access tier servers, there are no management attributes or operations exposed in the Administration Console. This configuration must be done directly on the MBeans using a JMX-based Administration Console such as JConsole or using WLST.

For example, if you are using WLST, connect to the MBean server as described in "WebLogic Scripting Tool (WSLT)" and change to the custom tree where Services Gatekeeper MBeans are located. Change to the Log4J directory: `cd('log4J')`.

The settings for the access tier servers can be changed in the following directories:

- log4j:appender=default
- log4j:appender=default,layout=org.apache.log4j.TTCCLayout
- log4j:hiearchy=default
- log4j:logger=rootdefault

For example, to change the trace level for the access tier servers for the appender log4j:appender=default to WARN:

```
cd('log4j:appender=default')
set('threshold','WARN')
```

Using the Log4J Configuration File

In addition to using the Log4J MBeans, trace can be configured using the file:

*Domain_Home***/log4j/log4jconfig.xml**

The file contains an empty skeleton, by default.

The settings defined in this file are read every 30 seconds.

The ordering of the elements must be taken into account. Appenders must be declared before categories.

When using appenders that write to a file, the files are stored in *domain_Home*.

Example Log4J Configuration file

Following is an example where the following named appenders are declared:

- MyRootLogger
- MyMultimediaMessagingWarningLogger
- MyAccountContainerLogger

All these appenders use the same class, `org.apache.log4j.FileAppender`.

Each appender writes to a file whose name is given in the `<param name="File" value="<File name>"/>` element. The appenders also use the same layout class, `org.apache.log4j.PatternLayout`, and ConversionPattern.

The appenders are used by a set of categories, where the name attribute defines which class to trace. The categories also define which Log4J priority that shall be logged.

In the example, the following services are logged:

- `com.bea.wlcp.wlng.account`, which is the Account service.
- `com.bea.wlcp.wlng.plugin.multimediamessaging`, which is the Parlay X Multimedia Messaging communication service.

Example 13–1 Example log4jconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">
<appender name="MyRootLogger" class="org.apache.log4j.FileAppender">
<param name="File" value="A1.log"/>
<param name="Append" value="false"/>
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%t %-5p %c{2} - %m%n"/>
</layout>
</appender>
<appender name="MyMultimediaMessagingWarningLogger" class="org.apache.log4j.FileAppender">
<param name="File" value="MMS_WARN.log"/>
<param name="Append" value="false"/>
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%t %-5p %c{2} - %m%n"/>
```

```
</layout>
</appender>
<appender name="MyAccountContainerLogger" class="org.apache.log4j.FileAppender">
<param name="File" value="Account.log"/>
<param name="Append" value="false"/>
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
</layout>
</appender>
<category name="org.apache.log4j.xml">
<priority value="info"/>
</category>
<!-- log for a certain service... -->
<category name="com.bea.wlcp.wlng.account">
<priority value="DEBUG"/>
<appender-ref ref="MyAccountContainerLogger"/>
</category>
<!-- end log for a certain service... -->
<!-- log for a certain service... -->
<category name="com.bea.wlcp.wlng.plugin.multimediamessaging">
<priority value="WARN"/>
<appender-ref ref="MyMultimediaMessagingWarningLogger"/>
</category>
<!-- end log for a certain service... -->
<root>
<priority value="debug"/>
<appender-ref ref="MyRootLogger"/>
</root>
</log4j:configuration>
```

TraceServiceMBean Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services** followed by **TraceService**. Alternately, use a Java application. For information on the methods and fields of the supported MBeans, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

14

Troubleshooting Your Services Gatekeeper Implementation

This chapter provides guidelines to help you troubleshoot problems with your Oracle Communications Services Gatekeeper implementation. You can find information about interpreting error messages, diagnosing common problems, and contacting Oracle customer support.

Before you read this chapter, you should be familiar with how Services Gatekeeper works. See *Services Gatekeeper Concepts* for information.

For information on problems related to Services Gatekeeper performance, see "Handling Performance Issues".

General Checklist for Resolving Problems with Services Gatekeeper

When any problems occur with your Services Gatekeeper system, it is best to do some troubleshooting before you contact Oracle:

- You know your installation better than Oracle does. You know if anything in the system has been changed, so you are more likely to know where to look first.
- Troubleshooting skills are important. Relying on Oracle to research and solve all of your problems prevents you from being in full control of your system.

Oracle needs a clear and concise description of the problem, including when it began to occur. If you have a problem with your Services Gatekeeper system, ask yourself these questions first, because Oracle will ask them of you:

- What exactly is the problem? Can you isolate it? For example, if users cannot authenticate, is it all services or just one service? Does it affect a specific NT server?
- Is this a known issue?

  Before calling to report an issue, it is a good idea to see if the problem you have encountered is a known issue already. Known issues are listed in *Services Gatekeeper Release Notes*.
- Do you have the log files?

  This is the first thing that Oracle will ask for. Check the error log for the Services Gatekeeper module with which you are having problems. Please keep the information handy when you contact Oracle. See "Using Error Logs to Troubleshoot Services Gatekeeper".
- Is the problem related to external systems?

Sometimes when there is an issue, the problem maybe related to the communication between Services Gatekeeper and an external system, such as a short message service center (SMSC) or a charging server.

This information is very helpful to Oracle in resolving such an issue. Capture the network traffic between Services Gatekeeper and the external system and provide it to Oracle.

- Have you read the documentation?

  Look through the list of common problems and their solutions in "Diagnosing Some Common Problems with Services Gatekeeper".

- Has anything changed in the system? Did you install any new hardware or new software? Did the network change in any way? Does the problem resemble another one you had previously? Has your system usage recently jumped significantly?
- Is the system otherwise operating normally? Has response time or the level of system resources changed? Are users complaining about additional or different problems?
- If the system appears completely dead, check the basics: Can you access the system administration console for Services Gatekeeper? Are other processes on this hardware functioning normally?
- Stay up-to-date (as much as possible) with the Services Gatekeeper patch set releases provided by Oracle.

  What is your current patch level? See "Finding the Current Patch Level of Your Services Gatekeeper System".

If the error message points to a configuration problem, check the configuration file for the associated module. If you find that the solution requires reconfiguring the module, change the configuration and verify if the problem was resolved.

If you still cannot resolve the problem, contact Oracle as described in "Getting Help for Problems with Services Gatekeeper".

Finding the Current Patch Level of Your Services Gatekeeper System

When you encounter an issue, then, before contacting Oracle, find the current patch level of your Services Gatekeeper system. Oracle will ask you for this information.

With the current patch level of your Services Gatekeeper system at hand, Oracle can tell you if your issue has been addressed in a later patch release. You can then easily solve the issue by upgrading your Services gatekeeper from its current level to that later patch level.

Listing What Is Currently Installed on Your Services Gatekeeper System

To list what is currently installed on your Services Gatekeeper system, use the **lsinventory** command from the OPatch utility.

OPatch is an Oracle-supplied utility that assists you with the process of applying interim patches to Oracle's software. The **lsInventory** command lists the inventory for a particular Oracle home, or displays all installations that can be found.

Running the OPatch lsinventory Command

To run the **lsinventory** command and obtain information on the patches that are applied currently on your Services Gatekeeper system:

Note: ORACLE_HOME needs to point to the installation on which opatch is to operate.

1. Go to the directory in which Services Gatekeeper is installed.

   ```
   cd <install_dir>
   ```

2. Set the WebLogic environment:

   ```
   source ./wlserver/server/bin/setWLSEnv.sh
   ```

3. Set ORACLE_HOME to the current directory

   ```
   export ORACLE_HOME=$(pwd)
   ```

4. Go to the subdirectory in which the OPatch utility resides.

   ```
   cd OPatch
   ```

5. Enter the following command.

   ```
   ./opatch lsinventory -detail
   ```

For more information on the **lsInventory** Command for OUI-based Oracle homes and OPatch, please see *Universal Installer and OPatch User's Guide* located on the Oracle Help center website.

Example 14–1 shows a sample output from the **lsInventory** command, when the command was used without the `-detail` parameter.:

Example 14–1 Sample Output from lsInventory

```
Oracle Interim Patch Installer version 13.2.0.0.0
Copyright (c) 2014, Oracle Corporation.  All rights reserved.


Oracle Home       : /home/username/oracle/ocsg_6.0_build_361
Central Inventory : /home/username/prog/oui_11.2.0.2.0
   from           : /home/username/oracle/ocsg_6.0_build_361/oraInst.loc
OPatch version    : 13.2.0.0.0
OUI version       : 13.2.0.0.0
Log file location : /home/username/oracle/ocsg_6.0_build_361/cfgtoollogs/opatch/opatch2014-10-30_
11-42-22AM_1.log


OPatch detects the Middleware Home as "/home/username/oracle/ocsg_6.0_build_361"

Oct 30, 2014 11:42:28 AM oracle.sysman.oii.oiii.OiiiInstallAreaControl initAreaControl
INFO: Install area Control created with access level  0
Lsinventory Output file location : /home/username/oracle/ocsg_6.0_build_
361/cfgtoollogs/opatch/lsinv/lsinventory2014-10-30_11-42-22AM.txt

--------------------------------------------------------------------------------

Interim patches (1) :

Patch  19836145     : applied on Thu Oct 30 11:25:24 CET 2014
Unique Patch ID:  1414504829609
Patch description:  "[Patch Set v6.0.0.1.7] - Patch bug for patch set XYZ"
   Created on 28 Oct 2014, 15:00:35 hrs PST8PDT
```

```
   Bugs fixed:
     123413, 123412



--------------------------------------------------------------------------------

OPatch succeeded.
```

Note the description of the patch as given in the above output, provides you with the patch set number (v6.0.0.1.7), that indicates you received the seventh update release for the first patch release of the 6.0 major release of Services Gatekeeper. It lists the numbers of the issues that were fixed.

Other Usages of the lsinventory Command

With the **lsinventory** command from OPatch, you can

- Group the inventory of all installed patches by the date they were installed in the Oracle home.

  ```
  ./opatch lsinventory -detail
  ```

- Pipe the output like any other command.

  ```
  ./opatch > out.log
  ```

- Redirect standard error (**stderr**) to standard output (**stdout**).

  ```
  ./opatch > out.log > 2>&1
  ```

Handling Performance Issues

Maintaining Services Gatekeeper performance levels is a complex task. If you find that your Services Gatekeeper system is not performing in an optimal manner, you may need to tune the underlying components to the requirements of your environment. For example:

- WebLogic Server

  If you find that your Services Gatekeeper system is not performing in an optimal manner, tune the underlying WebLogic Server (WLS) to the requirements of your environment. For example, select the appropriate startup mode for your installation.

  For information about the default tuning values for WebLogic Server development and production modes, see *Oracle Fusion Middleware Performance and Tuning for Oracle WebLogic Server*.

- Java Virtual Machine (JVM)

  How you tune your JVM affects the performance of WebLogic Server and your applications. For more information see the discussion on tuning Java Virtual Machines (JVMs) in *Fusion Middleware Performance and Tuning for Oracle WebLogic Server* on the Oracle Help Center website.

- Persistence type for storage services

  If you find that your Services Gatekeeper system is not performing in an optimal manner, check on the caching technique you have implemented. Compare the techniques to configure one that better suits your requirement to storing and accessing the data.

For example, the write-through caching technique has performance implications when compared to the write-behind technique. This is because, for write-through, the data input/output operation to cache and to the permanent storage location must complete first before a notification is sent to the host.

- Latency

  If you find that your Services Gatekeeper system is not performing in an optimal manner, check the network latency and network performance between the application tier and the database tier. See *Latency and Bandwidth Requirements* for information on the requirements that Oracle recommends.

  The traffic between your application and your database could be a factor, especially in a multi-tiered environment.

As part of your discovery process on Service Gatekeeper performance, be sure to look at the log files that Services Gatekeeper provides.

Diagnosing Problems from Alarms

If Services gatekeeper encounters a problem that it recognizes, it sends an EDR alarm to help you diagnose the problem. See "Managing and Configuring EDRs, CDRs and Alarms" for general information about alarms, and *Alarms Handling Guide* for details on the individual alarms organized by **tagalarm** number.

Using Error Logs to Troubleshoot Services Gatekeeper

If you are having a problem with Services Gatekeeper, look in the log files. Log files include errors that need to be managed, as well as errors that do not need immediate attention (for example, invalid logins).

To manage log files, you should make a list of the important errors for your system, as opposed to errors that do not need immediate attention.

About Error Log Files

Services Gatekeeper maintains a **default.log** file that contains logs from the modules specific to it. The error log files provide detailed information about system problems.

Additionally, look at the entries in the WLS server log files.

Finding Error Log Files

The Services Gatekeeper specific log file, **default.log** is located at:

```
domain_root_dir/servers/server_name/trace
```

The log files for the servers are located at:

```
domain_root_dir/servers/server_name/logs
```

By default, *domain_root_dir* represents the directory in which WebLogic Server domain is created and *server_name* is the name of the server.

Resolving Clusters of Error Messages

An error often produces a cluster of error messages in the log file. Some errors may tend to generate cascading messages. To resolve the error, try and locate the first one in the series.

Changing Log Levels in Services Gatekeeper

An easy and persistent way to change the logging level is to edit the **log4j** configuration file under *Domain_Home*/**log4j/log4jconfig.xml**.

To obtain a complete log, change the priority value:

1. Go to the directory where the **log4jconfig.xml** configuration file is located.

   By default, it is in the Services Gatekeeper domain at *Domain_Home*/**log4j**.

2. Open the **log4jconfig.xml** configuration file in an appropriate text editor.
3. Locate **priority value=** entry.
4. Set **priority value** to all, as shown below:

   ```
   <root>
           <priority value="all"/>
   </root>
   ```

5. Save the file.

Collecting Log Data

Generally, server logs are important. Collect log information while the entries are fresh. If the log files are rotated, then eventually old logs will be overwritten by new ones.

Here is an example of how to collect Services Gatekeeper and WebLogic Server logs from a node. Copy and save the appropriate script to your Services Gatekeeper installation directory. Run the script from the same directory, repeating it for all nodes.

Use the script in Example 14–2 for Linux installations.

Example 14–2   Example of a Script to Collect Logs (Linux)

```
Linux version
#!/bin/sh

#This will collect all log, out, configuration and recording files
ROOT=`pwd`
ARCHIVE_DIR=/tmp
MACHINE=`hostname`
echo $MACHINE
ARCHIVE_FILE=${ARCHIVE_DIR}/`date +%F_%H_%M_%S`
TMP_FILE_LIST=${ARCHIVE_DIR}/tarinput

find $ROOT | grep -e".*\.log[\.,0-9]*$" > $TMP_FILE_LIST
find $ROOT | grep -e".*\.jfr$" >> $TMP_FILE_LIST
find $ROOT | grep -e".*\.xml$" >> $TMP_FILE_LIST
find $ROOT | grep -e".*\.out$" >> $TMP_FILE_LIST
tar cvf ${ARCHIVE_FILE}_$MACHINE.tar -T $TMP_FILE_LIST
gzip ${ARCHIVE_FILE}_$MACHINE.tar
echo "Created archive ${ARCHIVE_FILE}_$MACHINE.tar.gz"
```

Use the script in Example 14–3 for Solaris installations.

Example 14–3 Example of a Script to Collect Logs (Solaris)

```
Solaris version
#!/bin/sh

#This will collect all log, out, configuration and recording files
ROOT=`pwd`
ARCHIVE_DIR=/tmp
MACHINE=`hostname`
echo $MACHINE
ARCHIVE_FILE=${ARCHIVE_DIR}/`date +%F_%H_%M_%S`
TMP_FILE_LIST=${ARCHIVE_DIR}/tarinput

find $ROOT | grep ".*\.log[\.,0-9]*$" > $TMP_FILE_LIST
find $ROOT | grep ".*\.jfr$" >> $TMP_FILE_LIST
find $ROOT | grep ".*\.xml$" >> $TMP_FILE_LIST
find $ROOT | grep ".*\.out$" >> $TMP_FILE_LIST
tar cvf ${ARCHIVE_FILE}_$MACHINE.tar -I $TMP_FILE_LIST
gzip ${ARCHIVE_FILE}_$MACHINE.tar
echo "Created archive ${ARCHIVE_FILE}_$MACHINE.tar.
```

Diagnosing Some Common Problems with Services Gatekeeper

This section describes some of the common problems you may encounter in Services gatekeeper. It shows you how to diagnose the error messages and resolve the following issues.

- Problem: The Server Will Not Start
- Problem: The Server is Hanging
- Problem: Memory Issues

Problem: The Server Will Not Start

The Services Gatekeeper server startup scripts work best with the Bash shell. If one of the server startup scripts fails with an error like this one:

```
./dbController.sh: 3: -/dbController.sh: Syntax Error: "(" unexpected
```

Edit the script, replacing the **#!/bin/sh** shebang with **#!/bin/bash**.

Problem: The Server is Hanging

A server (or node) may hang due to more than one reason.

When you find that a server is hanging, regardless of the actual cause, it is always a good idea to capture a thread dump while the node is hanging.

Note: To identify slow-moving threads, be sure to take two thread dumps thirty (30) seconds apart.

If you capture the thread dump before you restart the node, you may find it easier to understand the reason why the node hanged. To store the thread dump in a log file, do one of the following: you will need to either use node manager or start the nodes so that standard out and standard error is forwarded to a file.

- Use Node Manager log file

Node Manager is a WebLogic Server utility that enables you to start, shut down, and restart Administration Server and Managed Server instances from a remote location. Although Node Manager is optional, it is recommended if your WebLogic Server environment hosts applications with high availability requirements.

For more information, see the discussion on Log Files in *Oracle Fusion Middleware Node Manager Administrator's Guide for Oracle WebLogic Server*

- Start the servers so that standard out and standard error is forwarded to a file. Run the starting script in the following way:

  ```
  startScript.sh > out.log 2>&1
  ```

If you have more than one thread dump, the results can be correlated to see if the states of the threads change. Example 14–4 shows how you can get a full list of the processes using the ps command.

Example 14–4 Obtaining Two thread Dumps

```
ps -ef | grep -e".*ocsg.*weblogic\.Server$"

#Oracle HotSpot Virtual Machine to print threads using jcmd
jcmd <pid> Thread.print > thread-dump.log
#wait for 30 seconds and do another dump
jcmd <pid> Thread.print > thread-dump.log

#Any JVM (output ends up on stderr)
kill -QUIT <pid from ps output>
#wait for 30 seconds and do another dump
kill -QUIT <pid from ps output>
```

Problem: Memory Issues

Garbage collection (GC) could result in long pauses that might affect performance.

To look for long GC pauses that might affect performance, add the **-verbose:gc** flag to your start script (**setDomainEnv.sh** for WebLogic-based servers). Example 14–5 shows the output seen when an example server is running with this flag.

Example 14–5 Example Garbage Collection Entries

```
[GC 307767K->235359K(375296K), 0.0803370 secs]
[GC 311327K->235207K(377024K), 0.0777140 secs]
[GC 313671K->216031K(344512K), 0.0520790 secs]
[GC 294495K->218928K(376448K), 0.0493060 secs]
[GC 295472K->218713K(341952K), 0.0441110 secs]
```

You can use the output to monitor the GC pauses while running traffic.

Getting Help for Problems with Services Gatekeeper

If you cannot resolve your problems with Services Gatekeeper, contact Oracle.

Before You Contact Oracle

Problems can often be fixed simply by shutting down Services Gatekeeper and restarting the computer that the Services Gatekeeper system runs on. See "Starting, Stopping, and Administering Servers" in *Services Gatekeeper System Administrator's Guide*.

> **Note:** Oracle will ask you for the relevant log files and thread dumps to troubleshoot an issue.
>
> Therefore, before you shut down Services gatekeeper, be sure to obtain the relevant log files and thread dumps associated with the issue.

If that does not solve the problem, the first troubleshooting step is to look at the error log for the application or process that reported the problem. See "Using Error Logs to Troubleshoot Services Gatekeeper". Be sure to review "General Checklist for Resolving Problems with Services Gatekeeper" before reporting the problem to Oracle.

Reporting Problems

If "General Checklist for Resolving Problems with Services Gatekeeper" does not help you resolve the problem, write down the pertinent information:

- A clear and concise description of the problem, including when it began to occur.
- Relevant configuration files.
- Recent changes in your system, even if you do not think they are relevant.
- List of all Services Gatekeeper components and patches installed on your system.

When you are ready, report the problem to Oracle.

15

Using Cloud Monitoring with Multi-tier Services Gatekeeper

This chapter explains how to add the multi-tier Oracle Communications Services Gatekeeper to Oracle Enterprise Manager Cloud Control (Enterprise Manager) monitoring control. Using Cloud Control with the single-tier Services Gatekeeper is not supported.

About Monitoring Multi-tier Services Gatekeeper Using Enterprise Manager

You use Enterprise Manager to monitor applications and their individual servers and services. Enterprise Manager is an Oracle product that you purchase, install, and configure separately.

You use Enterprise Manager to monitor applications and their individual servers and services. Enterprise Manager allows you to monitor a Services Gatekeeper implementation as a whole, and as a collection of Enterprise Manager *targets* that you can configure and monitor individually. Each target generally corresponds to a Services Gatekeeper server or process. For example, you can monitor the state of a Services Gatekeeper NT server and an SMS process running on that server separately.

You can use Enterprise Manager to monitor Services Gatekeeper, but not to configure or manage it.

You can use the Enterprise Manager to:

- Monitor the running or stopped status of Services Gatekeeper.
- Monitor the running or stopped status of each target (server or service) individually.
- Retrieve and monitor statistics on the running Services Gatekeeper implementation.
- Retrieve and monitor statistics for each of the targets within a Services Gatekeeper implementation.
- Retrieve events and alarms.

See the discussion "about Enterprise Manager compatibility" in *Services Gatekeeper Multi-tier Installation Guide* for the version of Oracle Enterprise Manager that Services Gatekeeper supports.

Integrating Enterprise Manager with Multi-tier Services Gatekeeper

See "*Integrating Services Gatekeeper with Enterprise Manager*" in *Services Gatekeeper Integration Guide* for the following:

- Details on the components that Enterprise Manager monitors.
- Instructions on integrating Enterprise Manager with Services Gatekeeper
- Information on how to sue the Enterprise Manager monitoring metrics.

Part II

Services Gatekeeper Advanced Administration

This part provides advanced Services Gatekeeper configuration information.

Part II contains the following chapters:

- Managing and Configuring SLAs
- Managing and Configuring the Storage Service
- Configuring Network Node Heartbeats
- Deploying and Administering Communication Services
- Configuring and Managing Communication Service Traffic
- Upgrading and Redeploying Communication Services and Service Interceptors
- Managing and Configuring the Tier Routing Manager
- Charging and Integrating Billing
- Implementing Diameter Ro Online Charging
- Implementing Diameter Rf Offline Charging
- Managing and Configuring Native UCP Connections
- Managing and Configuring Parlay X 2.1 Shortcode Mappings
- Managing OSA/Parlay Gateway Connections Using Parlay_Access
- Managing Legacy Application Service Providers

16

Managing and Configuring SLAs

This chapter explains how to configure Oracle Communications Services Gatekeeper budgets and describes their relationship to Service Level Agreement (SLA) settings.

Understanding How the PRM Portals Use SLA Settings

Using the PRM Portals you create simple SLAs based on these parameters:

- Group limits:
  - Request Limit
  - Quota
- Application requests:
  - Request Limit
  - Quota

If these bandwidth settings are appropriate for your implementation, there is no need to use this chapter to create more sophisticated budgets. However, if your implementation requires more fine-grained access and bandwidth settings, you must use the instructions in this chapter to create SLAs.

> **Note:** Either use the PRM portal SLA settings, or SLAs that you create using the **BudgetServiceMBean** but not both. Both methods create SLAs and they will overwrite each other, creating unexpected results.

Understanding Budgets

In Services Gatekeeper, SLA enforcement is based on budgets maintained by the Budget service. Based on the rates, restrictions, and quotas set up in SLAs, budgets measure the level of access an application has to Services Gatekeeper over time. The budget is continuously being *decremented* based on the application's use of Services Gatekeeper. At the same time, the budget is being *incremented* based on the contract between the service provider and the operator. If the application's use of Services Gatekeeper exactly matches the SLA values, the budget level remains the same. The Platform Text Environment (PTE) includes a budget monitoring tool that tracks the state of budget usage over time

The budget reflects the current traffic request rate based on traffic history. Each Services Gatekeeper server updates both its own local traffic count and the clusterwide count maintained in one Services Gatekeeper server, the cluster master, based on load and time intervals. The cluster master is, from a cluster perspective, a singleton service that is highly available and is guaranteed to be available by the WebLogic Server infrastructure. The cluster master is also guaranteed to be active on only one server in the cluster. This ensures accurate SLA enforcement with regards to request counters.

By default, budget quotas are enforced within the cluster. The Budget service is also capable of maintaining budget quotas across domains spread across geographic locations.

Budget values for SLAs that span longer periods of time are persisted in the persistent store to minimize the state loss if a cluster master fails.

There are two types of budget caches:

- In-memory only
- In-memory cache backed by persistent storage

When a cluster master is restarted, it revives its state from the persistent store. If a cluster master fails, each Services Gatekeeper server continues to independently enforce the SLA accurately to the extent possible, until the role of cluster master has been transferred to an operational server. In such a situation, a subset of the budget cache is lost: the in-memory-only budget cache and the parts of the in-memory cache backed by persistent storage that have not been flushed to persistent storage. The flush intervals are configurable using the **PersistentBudgetFlushInterval** and **PersistentBudgetTimeThreshold** fields to **BudgetServiceMBean**.

The configuration settings for the following attributes of the **BudgetServiceMBean** MBean affect accuracy and performance:

- The higher the value for the **AccuracyFactor** field, the more granularity you have in enforcing the budgets over the time span. This requires more processing power to synchronize the budgets over the cluster.
- The higher the value for the **PersistentBudgetFlushInterval** field, the less impact persisting data has on overall performance, and the more budget data may be lost in case of server failure.
- The higher the value for the **PersistentBudgetTimeThreshold** field, the fewer budgets are likely to be persisted. This value is related to the time intervals for which time limits are defined in the SLAs. A high threshold causes less impact on database performance, but more data may be lost in case of server failure.

For information about using **BudgetServiceMBean** see "BudgetServiceMBean Reference."

Synchronizing Budgets Between Servers

Budgets are synchronized between all servers in a cluster according to the following algorithm:

`rt = r / (a * n)`

`Tt = T / (a * n)`where:

- **rt** is the slave request count synchronization threshold value.
- **r** is a request limit specified in an SLA.
- **a** is the accuracy factor; use the **AccuracyFactor** field
- **n** is the number of running WebLogic Network Servers in a cluster.

- **Tt** is the duration of counter synchronization between the slave and the master.
- **T** is a time period specified in the SLA.

Understanding Slave intervals

The request count is the amount of the budget that has been allocated since the last synchronization with the master. The following scenarios are possible:

1. When the request count reaches r_t on a particular node, it synchronizes with the master.
2. If the request count does not reach the r_t value and if the count is greater than zero, the slave synchronizes with the master if the time since last synchronization reaches T_t.

Synchronization happens as a result of (1) or (2), whichever comes first.

If the request count reaches the threshold value, there will be no explicit synchronization when the timer reaches T_t.

Example:

If r = 100, n = 2 and T = 1000 milliseconds and a = 2

r_t = 100 / (2 * 2) = 25 requests T_t = 1000 / (2 *2) = 250 milliseconds

The slave synchronizes with the master if the request count reaches 25 or if the time since the last synchronization is 250 milliseconds, whichever comes first, at which point the timer is reset.

Understanding Masters

The master is responsible for enforcing the budget limits across the cluster by keeping track of the request count across all the servers in the cluster. If there is budget available, the master updates the slaves with the remaining budget whenever the slaves synchronize with the master.

Understanding Failure Conditions

In the absence of the master, each slave individually enforces the budget limit but caps the requests at r/n, thereby guaranteeing that the budget count never reaches the limit.

If the slave fails before it can update the master, the master is not able to account for that server’s budget allocation and can be r_t requests out of sync.

Under certain circumstances, if the master allocates more than the configured budget limit, the budget will be adjusted over time.

For budgets that span longer periods of time, the budget count is persisted in the database to avoid losing all state during master failures. Use the **PersistentBudgetTimeThreshold** field of **BudgetServiceMBean** for this.

For information about using **BudgetServiceMBean** see "BudgetServiceMBean Reference."

Understanding Budget Overrides

Budgets can have overrides defined in the SLAs. When a budget is configured with an override, the budget master determines if a given override is active. If an override is active, the budget master enforces limits based on that active override configuration. If overrides are overlapping, no guarantees are provided on which override will be enforced.

Note: For an override to be active, all of the following must be true:

- Today's date must be the same as or later than `startDate`.
- Today's date must be earlier than `endDate` (must not be the same date).
- Current time must be between `startTime` and `endTime`. If `endTime` is earlier than `startTime`, the limit spans midnight.
- Current day of week must be between `startDow` and `endDow` or equal to `startDow` or `endDow`. If `endDow` is less than `startDow` the limit spans the week end.

Overrides are not enforced across geographically redundant sites.

Budget Calculations and Relationship to SLA Settings

Budgets are calculated based on the following SLA settings:

- <reqLimit> and <timePeriod> for request limits.
- <qtaLimit> and <days> for quotas.
- <reqLimitGuarantee> and <timePeriodGuarantee> for guarantee settings

The limits divided by the time-period translates into a budget increase rate, expressed in transactions per second.

Each budget has a maximum value define in the limits (<reqLimit>, <qtaLimit, and <reqLimitGuarantee>). This value is also the starting value of the budget. For each requests that is processed, the budget is decreased with one (1). Over time, the budget increases with the budget increase rate multiplied with the time, and its maximum value is the request limit. When a budget has reached the value of zero (0), requests are denied.

A maximum request rate is expressed as the number of request during a a time-period, so it offers very flexible ways to define the limits. For a given budget increase rate, expressed in transactions per second:

- The longer time-period defined, the longer it takes for requests to start to be rejected if the request rate is higher than allowed since the maximum budget value is higher.
- The shorter time-period defined, the sooner requests are starting to be rejected if the request rate is higher than allowed since the maximum budget value is lower.

Having a shorter time period means that budget synchronizations are more frequent and this has a performance impact.

If an application has used up its budget by sending more requests than allowed, and requests have started to be rejected, and the application reduces the request rate, the budget starts to increase. The budget is increased with the delta between the budget increase rate and the request rate.

Example:

An application sends 250 requests per second.

The SLA settings are:

<reqLimit>2000</reqLimit>

<timePeriod>10000</timePeriod>

This means that the budget increase rate is 2000/10000= 200 requests per second.

The difference between the request rate and the budget increase factor is 50 (250-200) so the budget will be zero (0) after 40 seconds (2000/50). When the budget is zero, requests are rejected.

Example:

An application sends 250 requests per second.

The SLA settings are

<reqLimit>200</reqLimit>

<timePeriod>1000</timePeriod>

This means that the budget increase rate is 200/1000= 200 requests per second.

The difference between the request rate and the budget increase rate is 50 (250-200) so the budget will be zero (0) after 4 seconds (200/50). When the budget is zero, requests are rejected.

Example:

An application sends 180 requests per second.

The SLA settings are

<reqLimit>200</reqLimit>

<timePeriod>1000</timePeriod>

Also, the current budget is zero (0) since it previously had a request rate that was higher than the allowed.

This means that the budget increase rate is 200/1000= 200 requests per second.

The difference between the budget increase rate and the request rate is 20 (200-180) so the budget will be at its maximum value (200) after 10 seconds (200/20). When the budget reaches its maximum value it does not increase any further.

Extending SLAs for Budget Services

By default, Services Gatekeeper uses plugin interface names and method names to control the budgets for services. You can fine-tune the rates, restrictions, and quotas set up in SLAs. To extend SLAs in Services Gatekeeper, you add optional elements and attributes to the XSD (XML schema definition).

The SLA shown in Example 16–1 contains a rate for the doStuff method of a plugin called DummyPlugInterface. It uses the <scs> element to specify the interface and service usage restrictions (rate and quota) in a <methodRestriction> element

Example 16–1 SLA Extended for an Example plug-in

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<Sla applicationGroupID="default_app_group"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation='app_sla_file.xsd'>
   <serviceContract>
      <startDate>2005-07-22</startDate>
      <endDate>9999-12-31</endDate>
      <scs>com.bea.wlcp.wlng.plugin.DummyPluginInterface</scs>
      <contract>
        <methodRestrictions>
           <methodRestriction>
```

```
                    <methodName>doStuff</methodName>
                    <rate>
                      <reqLimit>1</reqLimit>
                      <timePeriod>5000</timePeriod>
                    </rate>
                    <quota>
                       <qtaLimit>1</qtaLimit>
                       <days>1</days>
                       <limitExceedOK>true</limitExceedOK>
                    </quota>
                 </methodRestriction>
             </methodRestrictions>
            </contract>
         </serviceContract>
      </Sla>;
```

When you use the dynamic API framework, incoming HTTP requests for a service are processed using logic defined in a generic servlet set up for the service. You create an XML file in which you expose your communication services as APIs and, in each case, provide their URLs and service usage details in the SLA associated with each application group. Each <scs> element contains the URL to access an API. The <methodRestriction> and <methodAccess> elements in the <contract> section provide the details about setup of the API for the service.

The SLA shown in Example 16–2 defines a OneAPI SMS (REST) service for the dynamic API framework.

Example 16–2 Example Service SLA in XML for Dynamic API Framework

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<Sla applicationGroupID="default_app_group" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation='app_sla_file.xsd'>
   <serviceContract>
      <startDate>2005-07-22</startDate>
      <endDate>9999-12-31</endDate>
      <scs>/1/smsmessaging/outbound/(.*)/sendMessage</scs>  <!--New value-->
      <contract>
        <methodRestrictions>
           <methodRestriction>
              <methodName>POST</methodName>  <!--New value-->
              <rate>
                <reqLimit>5</reqLimit>
                <timePeriod>1000</timePeriod>
              </rate>
              <quota>
                 <qtaLimit>1</qtaLimit>
                 <days>1</days>
                 <limitExceedOK>true</limitExceedOK>
              </quota>
           </methodRestriction>
       </methodRestrictions>
       <methodAccess>
          <blacklistedMethod>
            <methodName>PUT</methodName>
          </blacklistedMethod>
          <blacklistedMethod>
            <methodName>HEAD</methodName>
          </blacklistedMethod>
       </methodAccess>
      </contract>
   </serviceContract>
   <serviceTypeContract>
        <serviceTypeName>/1/smsmessaging/</serviceTypeName>
        <startDate>2012-07-31+08:00</startDate>
        <endDate>2018-12-31+08:00</endDate>
        <rate>
            <reqLimit>200</reqLimit>
            <timePeriod>1</timePeriod>
        </rate>
    </serviceTypeContract>
    <composedServiceContract>
        <composedServiceName>ComposedService</composedServiceName>
        <service>
           <serviceTypeName>/1/smsmessaging/</serviceTypeName>
        </service>
        <service>
           <serviceTypeName>/1/mmsmessaging/</serviceTypeName>
        </service>
        <startDate>2012-07-31+08:00</startDate>
        <endDate>2018-12-31+08:00</endDate>
        <rate>
            <reqLimit>400</reqLimit>
            <timePeriod>1</timePeriod>
        </rate>
    </serviceTypeContract>
</Sla>;
```

As Example 16–2 shows, the <scs> element contains the request url for the SendMessage request API for the Restful service. The <methodRestriction> element specifies the details of the service quota and limits for the supported HTTP method, POST. The HTTP methods PUT, HEAD, and DELETE are not supported and therefore are blacklisted within the <blacklistedMethod> element under the <methodAccess> element.

Configuring and Managing Budgets

Configure the following fields for the **BudgetServiceMBean**:

- **PersistentBudgetFlushInterval**
- **PersistentBudgetTimeThreshold**
- **AccuracyFactor**
- **ConfigUpdateInterval**

No management methods are available.

For information about using **BudgetServiceMBean** see "BudgetServiceMBean Reference."

BudgetServiceMBean Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services** followed by **BudgetService**. Alternately, use a Java application. For information on the methods and fields of the supported MBeans, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Adding a Datasource

A *datasource* is an abstraction that handles connections with the persistent store. Under normal operating conditions, Services Gatekeeper, including the Budget service and the WebLogic Server automatic migration framework, share the common transactional (XA) datasource (**wlng.datasource**) that has been set up for Services Gatekeeper at large.Under very heavy traffic, it is possible for the Budget singleton service to be deactivated on all servers. This can happen if the automatic migration mechanism that supports the service becomes starved for connections. In this case, a major severity alarm is thrown: Alarm ID 111002, "Budget master unreachable". Although datasource issues are not the only reason you might receive this alarm.

If you encounter this problem, you can set up a separate singleton datasource for the migration mechanism to assure that the singleton service always has access to the persistent store. This datasource should be configured to use wlng.datasource. For more information on automatic migration of singleton services, see "Automatic Migration of User-Defined Singleton Services" in *Oracle WebLogic Server Administering Clusters for Oracle WebLogic Server.*

Also see the section about high-availability database leasing in that document. This is the mechanism underlying migration.

For information on setting up a separate datasource to support migration of singleton services, like the Budget service, see "Configuring JDBC Data Sources" in *Oracle Fusion Middleware Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*.

17

Managing and Configuring the Storage Service

This chapter describes how to configure and manage the Oracle Communications Services Gatekeeper Storage service that communication services use to store data.

Understanding the Storage Service

Services Gatekeeper Storage service provides transparent data access to communication services and container services. Conceptually, a module uses the Storage service, which in turn uses one or more underlying storage providers that define how and where data is stored. A module creates and uses one or more named stores which can be of different store types. The current storage provider is the Invalidating Storage Provider, an in-memory cache backed by a persistence store, currently a database, acting as the master. The store is a write-through cache. In order to maintain a coherent view of the cache, invalidating events are broadcast within the cluster for specific operations.

Table 17–1 outlines the store types.

Table 17–1 Services Gatekeeper Store Types

Type	Description
Cluster cache	A clusterwide in-memory-only store, ideally with redundancy support by keeping backup copies on one, or more of the other servers in the cluster. Because this is an in-memory only store, data reads/writes are very fast.
Write-behind database store	A clusterwide in-memory cache, ideally with redundancy support implemented by keeping backup copies on one or more of the other servers in the cluster and also backed by a database. Updates for the database table for write-behind stores are delayed and asynchronously written in batches to the database. This store has similar performance characteristics to the cluster store for data that is available in the cache, but with better availability.
Write-through database store	A clusterwide in-memory cache of a database table. Updates for the database table for write through stores are done synchronously as part of the store update operation. Updates to data in the store are slow compared to the cluster store, but read operations are fast if the data is available in cache. This store offers best reliability.

Table 17–1 (Cont.) Services Gatekeeper Store Types

Type	Description
Refresh ahead database store	This option configures a cache to automatically and asynchronously reload (refresh) any recently accessed cache entry from the cache loader before its expiration. The result is that after a frequently accessed entry has entered the cache, the application does not feel the impact of a read against a potentially slow cache store when the entry is reloaded due to expiration. The asynchronous refresh is only triggered when an object that is sufficiently close to its expiration time is accessed. If the object is accessed after its expiration time, Coherence performs a synchronous read from the cache store to refresh its value.
Database log store	The log store type data updates to the database table are delayed and asynchronously written in batches to the database. This type of store is meant for additive batch writing of write only data. Because this type of store is not meant for reading, updating or deleting data, it does not need to keep a cache. Because this store is meant only for adding data, read performance is poor. However, it is optimized for add operations by performing database writes in batches.

Each communication service has its own store definition in the directory

domain_home/**config/store_schema**.

Where *domain_home* is the directory in which the Services Gatekeeper domain resides, located in *Middleware_home*/**user_projects/domains**.

The store definitions are JAR files with a configuration file and class definitions for the stored data.

Configuring Storage Expiration

To prevent the databases underlying the Storage service from growing too large, you can set an expiration date on Storage service data, after which time the data is deleted. One consequence is that a delivery receipt can be unsuccessful if a message is so old that gets deleted.

You can configure both the data expiration period for a communication service and how often that Services Gatekeeper checks for expired data. See "Configuring the System Wide Expiration Interval" for details on setting system wide expiration limits.

You also have the option of overriding the system wide expiration settings for individual plug-ins or communication services. When the NT servers starts, it probes the system for store expiration settings in this order:

1. First it probes whether an XML file exists with the same name as a store jar file. If it finds one, it uses any settings in that file. For example, **com.bea.wlcp.wlng.cc_interceptor.store_6.0.0.0.xml**.
2. If no XML file is found, the NT server probes for an internal configuration file, and uses any settings in it. See "Overriding the Expiration Interval Using Jar files" for details on using an internal configuration file.
3. Finally, the NT server probes for an expiration setting in the **store.properties** file. For details on using the **store.properties** file, see "Overriding the Expiration Interval Using the store.properties File".

Configuring the System Wide Expiration Interval

At the system level, Services Gatekeeper checks whether storage data has expired at a set interval defined by the **com.bea.wlcp.wlng.storage.expiry_period** variable. The default for the **expiry_period** is 1800000 milliseconds (thirty minutes).

You can configure this value on server startup by passing it as a Java option when you restart Services Gatekeeper. The option is:

```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dcom.bea.wlcp.wlng.storage.expiry_
period=expiry_period_value"
```

For example, the following flag configures Services Gatekeeper to check for expired data once an hour:

```
JAVA_OPTIONS="${JAVA_OPTIONS} -Dcom.bea.wlcp.wlng.storage.expiry_
period=3600000"
```

Overriding the Expiration Interval Using Jar files

This section explains how to override the system wide expiration interval for an individual plug-in or communication service using the jar file.

> **Note:** Perform this task whenever the jar file is replaced.

To override the system wide expiration interval:

1. Locate the store schema JAR file for the store that you want to configure.

   These jar files are in under the *domain_home***/config/store_schema** directory.

   For example, the JAR file for the common SMPP store is *domain_home***/config/store_schema/com.bea.wlcp.wlng.sms.common.store**.

2. Extract the files from the JAR file for the store that you want to configure.
3. Open the **wlng-cachestore-config-extensions.xml** file.
4. In the extensions file, create or edit the `<expiry>` element, setting its `expiry_age` value. The value is in seconds.
5. Save the file.

If `expiry_age` is not set, the data never expires.

The following example shows the configuration of the MO message store setting the expiration period to 604800 seconds (one week).

```
<!-- MO message store (common) start -->
  <db_table name="pl_sms_mo_sms">
    <key_column name="msg_id" data_type="VARCHAR(30)"/>
    <bucket_column name="smsdata" data_type="BLOB"/>

    <value_column name="notif_correlator" data_type="VARCHAR(30)">
      <methods>
        <get_method name="getNotificationInfoCorrelator"/>
        <set_method name="setNotificationInfoCorrelator"/>
      </methods>
    </value_column>
    <expiry expiry_age="604800"/> <!-- 1 week  -->
  </db_table>
```

Overriding the Expiration Interval Using the store.properties File

This section explains how to override the system wide expiration interval for individual plug-ins or communication services using a **store.properties** file that you create. The steps that restart the servers ensure that your changes are propagated to your other servers.

These changes are persistent, and need not be repeated if the plug-in jar files are replaced.

To override the system wide expiration interval:

1. Navigate to *domain_home*/**config/store_schema**.
2. Create a **store.properties** file using any text editor.
3. Add an entry to set the expiration interval (in seconds) for a plug-in with this syntax:

   *table_name***.expiry=***expiration_interval*

   For example, this entry sets the SMS Plug-in expiration interval to five minutes:

   pl_sms_smpp_mt_sms.expiry=300
4. Repeat step 3 for all plug-ins or communication services that you are overridign the system expiration interval for.
5. Save and close **store.properties**.
6. Restart the administration server.
7. (In a multi-tier implementation) Restart the network tier server.

StorageServiceMBean Reference

Set field values and use methods from the Administration Console, by selecting **Container**, then **Services**, and then **StoreageServiceMBean**. Alternately, use a Java application. For information on the methods and fields, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

18

Configuring Network Node Heartbeats

This chapter describes how to configure heartbeats for HTTP-based stateless network protocol plug-ins for Oracle Communications Services Gatekeeper.

Understanding Network Node Heartbeats

The heartbeat functionality performs heartbeat checks on HTTP-based network nodes on behalf of a Services Gatekeeper communication service plug-in. When a heartbeat fails, the plug-in is set to an INACTIVE state. The heartbeat functionality continues trying to connect to the node and, when a positive answer is received, the plug-in re-enters the ACTIVE state.

The following network protocol plug-ins use this functionality:

- Parlay X 2.1 Terminal Location/MLP
- Extended Web Services WAP Push/PAP
- Parlay X 2.1 Multimedia Messaging/MM7
- Native MM7

Configuring and Managing Heatbeats

The heartbeat operation administration and management (OAM) WebLogic interface functionality is shared among the plug-ins. They all use the same MBean, **com.bea.wlcp.wlng.heartbeat.management.HeartbeatMBean**. But the result is rendered per plug-in (one instance is displayed per plug-in). It appears slightly differently depending on how to the MBean is accessed:

- In the Administrative Console, expand the plug-in that uses heartbeat functionality, for example:
  - Plugin_px21_multimedia_messaging-HeartbeatConfiguration
  - Plugin_ews_push_message_papHeartbeatConfiguration
  - Plugin_px21_terminal_location_mlp-HeartbeatConfiguration
  - Plugin_multimedia_messaging_mm7-HeartbeatConfiguration

  Then click **HeartBeat Configuration** to display the attributes for the heartbeat module.
- In an MBean browser, such as JConsole, one instance of the MBean is displayed using the same ObjectName (at the same hierarchical level) as the plug-in it is used by.

Note: You must configure heartbeat attributes for *all* of the above mentioned plug-ins that use this feature. Only the heartbeat attributes for the related plug-in are displayed in the console.

To configure heartbeat attributes:

1. Specify if heartbeats should be enabled for the associated plug-in using the **HeatBeatMbean Enabled** field.
2. If heartbeats are enabled, set values for these **HeartBeatMBean** fields:
   - **ContentMatch**
   - **Interval**
   - **NetworkServiceUrl**

HeartBeatMBean Reference

Set field values and use methods from the Administration Console (**Container Services** > **HeartbeatMBean**) or a Java application. For information on the methods and fields, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

19

Deploying and Administering Communication Services

This chapter describes how to deploy and administer communication services in Oracle Communications Services Gatekeeper.

About Communication Services

All telecommunication traffic processed by Services Gatekeeper passes through a communication service. Each communication service includes an application-facing (north) interface connecting it to an application, a service plug-in that provides functionality, and a network-facing (south) interface connecting it to your network nodes.

For:

- An overview of communication services, see *Services Gatekeeper Concepts*.
- Details on the communication services that Services Gatekeeper includes, see *Services Gatekeeper Communication Service Reference Guide*.
- Information on creating your own custom communication services see *Services Gatekeeper Extension Developer's Guide*.

Understanding How Communication Services are Packaged

Communication services are packaged and deployed in EAR files. Container services are packaged and deployed separately from the communication services and should not be modified.

All network protocol plug-ins that share the same group of application-facing interfaces are packaged into the same EAR file.

All communication services that share a common set of application-facing interfaces are grouped together. For example, Services Gatekeeper is delivered with two communication services for Parlay X 2. 1 Third Party Call:

- Parlay X 2.1 Third Party Call/INAP, where INAP is exposed through Parlay X 2.1 Third Party Call interfaces.
- Parlay X 2.1 Third Party Call/SIP, where SIP is exposed through Parlay X 2.1 Third Party Call interfaces.

Each group of communication services is packaged in two separate EAR files:

- **wlng_at_***communication service***.ear**: This file serves as the service facade for the named communication service. It consists of modules shared among the communication service only. The **wlng_at_***communication service***.ear** file is deployed in the access tier.

- **wlng_nt_***communication service***.ear**. This file serves as the service enabler for the named communication service. It contains the network protocol plug-ins for the communication service and common modules for the communication service. The **wlng_nt_***communication service***.ear** file is deployed in the network tier.

The communication services EAR files are located in the *Middleware_home***/ocsg_ /applications** directory.

A Communication Service Packaging Example

The files holding the communication services that share the Parlay X 2.1 Third Party Call communication service layer consist of the following artifacts:

- **wlng_at_third_party_call_px21.ear**
- **wlng_nt_third_party_call_px21.ear**

About the wlng_nt_third_party_call_px21.ear File

The **wlng_nt_third_party_call_px21.ear** file contains, among other modules:

- **Plugin_px21_third_party_call_inap.jar**, which contains the Parlay X 2.1 Third Party Call/INAP plug-in.
- **Plugin_px21_third_party_call_sip.jar**, which contains the Parlay X 2.1 Third Party Call/SIP plug-in.

The **Plugin_px21_third_party_call_sip.jar** file is one of the artifacts needed to achieve end-to-end service communication for communications services connecting to the SIP network. There is also a part that is deployed as a SIP servlet in the SIP Server container. The SIP servlet parts are packaged in a set of files, **wlng-integration-management.jar**, **wlng-integration-console-ext.war**, **wlss-int.ear**, and **wlng-security.jar**.

An HTTP servlet is available for plug-ins that use HTPP as the transport protocol and handle network-triggered messages from the network node. HTTP servlets are packaged as Web Archives (WARs) and are packaged in the network tier EAR for the communication service.

Other modules in **wlng_nt_third_party_call_px21.ear** are shared between the two network protocol plug-ins, including the common parts that are tied to the implementation of the communication layer, and any libraries and utilities shared among the plug-ins.

When adding or removing a plug-in to or from **wlng_nt_***communication service***.ear**, expand the EAR and the plug-in specific parts that are being added or removed.

Example 19–1 describes the elements in a **wlng_nt_***communication service***.ear** file.

Example 19–1 Contents of wlng_nt_<communication service>.ear

```
+---APP-INF/lib
...
communication service_callback_client.jar
/<utilities 1>.jar
...
/<utilities n>.jar
+---META-INF/MANIFEST.MF
/application.xml
```

```
/weblogic-application.xml
/weblogic-extension.xml
+---<plug-in 1>.jar
...
+---<plug-in n>.jar
+---<plug-in 1>.war
...
+---<plug-in n>.war
+---<communication service>_service.jar
```

In Example 19–1:

- **APP-INF/lib** contains any JARs that are shared among the plug-ins in the EAR. This includes the client library for the service callback EJB *communication service_* **callback_client.jar**.

  One or more utility jar files can be present, depending on the type of communication service.

- **META-INF/MANIFEST.MF** is a standard manifest file.
- **META-INF/application.xml** is the standard deployment descriptor for EARs.
- **META-INF/weblogic-application.xml** is the WebLogic Server-specific deployment descriptor.
- **META-INF/weblogic-extension.xml** is a WebLogic Server-specific deployment descriptor.

All plug-ins in the service enabler are packaged as individual jar files in the root of the EAR with the service EJB, *communication service*_**service.jar**.

If a plug-in connects to the telecom network using HTTP and supports network-triggered requests, there is also a corresponding WAR file that contains the servlet.

For more information about enterprise application deployment descriptor elements, see “Referencing JMS Resources in a WebLogic Application” in *Oracle WebLogic Server Developing Applications for Oracle WebLogic Server*.

Example 19–2 describes the elements in a plug-in jar file.

Example 19–2 Contents of <plug-in X>.jar

```
+---com/
+---org/
+---META-INF/MANIFEST.MF
+   instancemap
+   srv_depl.xml
```

The plug-in jar file contains the regular elements in a jar and an `instancemap`. The `instancemap` element uses the **InstanceFactory** class to instantiate the plug-in specific implementation of the required interface. See the discussion about "Retrieving Implementation Instances Using InstanceFactory" in *Services Gatekeeper Extension Developer's Guide* for more information.

Deploying SOAP and RESTful Facades on Multiple AT Clusters

For those communication services that have RESTful facades, both SOAP and RESTful facades are included in the standard EAR files.

If your installation needs to use multiple clusters of AT servers, you must make some adjustments. You cannot deploy the RESTful facade on multiple clusters within the same domain. You can deploy both the RESTful facade and the SOAP facade on a single cluster within the domain and deploy only the SOAP facade on multiple other clusters within that same domain.

To accomplish this, do the following:

1. Undeploy the standard EAR files for the communication services (including the session manager) you are using on any additional clusters your installation requires.
2. Replace the files with special equivalent EAR versions that contain only SOAP interfaces

The file names for **.ear** files for SOAP interfaces are identical to those of the standard files except that a **_soap** is appended to the end of the name. The following SOAP-only EAR files should be deployed (as needed) in this situation:

- **wlng_at_call_notification_px21_soap.ear**
- **wlng_at_multimedia_messaging_px21_soap.ear**
- **wlng_at_payment_px30_soap.ear**
- **wlng_at_presence_px21_soap.ear**
- **wlng_at_push_message_ews_soap.ear**
- **wlng_at_session_soap.ear**
- **wlng_at_sms_px21_soap.ear**
- **wlng_at_subscriber_profile_ews_soap.ear**
- **wlng_at_terminal_location_px21_soap.ear**
- **wlng_at_third_party_call_px21_soap.ea**r

These files are found in the *Middleware_home***/ocsg/applications** directory of your installation. Information on the content of these files is available in the “Properties” section of each respective communication services reference in *Services Gatekeeper Communication Service Reference Guide*.

About the Deployment Procedure

Complete the deployment procedure before production.

Important:

- Excluding step 6, Oracle recommends that you make the console-based changes with only the administration server running.
- For step 6, you must shut down all servers and restart.

From the Administration console:

1. Create Managed Servers for the non-REST AT clusters (for example, WLNG_AT1 and WLNG_AT2).
2. Create Managed Servers for the REST_AT cluster (for example, REST_AT1 and REST_AT2)

3. Create the clusters for these servers (for example WLNG_AT_Cluster and REST_AT_Cluster).
4. Assign the Managed Servers to their appropriate clusters:
   - WLNG_AT_Cluster WLNG_AT1 WLNG_AT2
   - REST_AT_Cluster REST_AT1 REST_AT2
5. Change all deployed applications with REST interfaces (that is, the standard EARs), including session manager, to target REST_AT_Cluster.
6. Change the targets of the AT JMS Servers from WLNG_AT* to the REST_AT*.
7. Change the target of WLNG_ATJMSResource from WLNG_AT_Cluster to REST_AT_Cluster.
8. Shut down all servers.

   Edit the following snippet of the **config.xml** file manually. In default installations this file can be found at *domain-home***/config/config.xml**.

   ```
   <custom-resource>
     <name>accesstier</name>
     <target>WLNG_AT_Cluster</target>
     <descriptor-file-name>custom/at.xml</descriptor-file-name>

   <resource-class>com.bea.wlcp.wlng.management.descriptor.resource.WlngTierResour
   ce</resource-class>

   <resource-class>com.bea.wlcp.wlng.management.descriptor.resource.WlngTierResour
   ce</resource-class>
   </custom-resource>
   ```

   Change the target in the access tier custom resource from WLNG_AT_Cluster to WLNG_AT_Cluster, REST_AT_Cluster.
9. Locate and open the **config.xml** file.

   In default installations, this file can be found at *domain-home***/config/config.xml**.
10. Carefully edit the following snippet of the **config.xml** file manually.

    ```
    <custom-resource>
      <name>accesstier</name>
      <target>WLNG_AT_Cluster</target>
      <descriptor-file-name>custom/at.xml</descriptor-file-name>

    <resource-class>com.bea.wlcp.wlng.management.descriptor.resource.WlngTierResour
    ce</resource-class>

    <resource-class>com.bea.wlcp.wlng.management.descriptor.resource.WlngTierResour
    ce</resource-class>
    </custom-resource>
    ```

    Change the target in the access tier custom resource from WLNG_AT_Cluster to WLNG_AT_Cluster, REST_AT_Cluster.
11. Start up the administration server and install and deploy the SOAP-only EAR files to the non-REST cluster (WLNG_AT_Cluster).
12. Start the newly installed EAR files using the Administration console. If you forget to do this, their status will be `prepared`.

13. Start the Managed Servers.

For more information on using the Administration Console to accomplish these tasks, click **Help** on the console screen.

Understanding Communication Service Version Handling

Communication services are associated with release versions and can be upgraded using in-production deployment.

The version numbering is on the EAR level, which means that all network protocol plug-ins for a given collection of application-facing interface are redeployed.

The version number for a specific *communication service* is specified in the `Weblogic-Application-Version` attribute in **META-INF/manifest.mf in wlng_at_***communication service***.ear** and **wlng_nt_***communication service***.ear**, respectively.

For more information about how to develop applications for production redeployment, see “Developing Applications for Production Redeployment" in *Oracle Fusion Middelware Developing Applications for Oracle WebLogic Server*.

Deploying and Undeploying Communication Services and Plug-ins

Communication services are deployed and undeployed as EAR files.

For a description of the different deployment options, see "Overview of the Deployment Process" in *Oracle Fusion Middleware Understanding Oracle WebLogic Server*.

The EAR file names for each communication service and the JAR names for the network protocol plug-ins are listed a table in the discussion on properties for each communication service chapter in *Services Gatekeeper Communication Service Reference Guide*.

The properties section also describes the JAR file for the plug-in and other artifacts, such as third-party libraries, used by the plug-in.

Following is an example on how to undeploy the Parlay X 2.1 Third Party Call communication service:

```
java weblogic.Deployer -adminurl http://<admin host>:<admin port> -user <admin
user> -password <admin password -name wlng_nt_third_party_call_px21 -undeploy
-graceful
```

If a plug-in has been removed from the EAR, use the mechanism described in "Performing a Hitless Upgrade".

Version Handling and Patching of Communication Services

See "Patch Management of Services Gatekeeper Systems" in *Multi-tier Installation Guide* for instructions on patching Services Gatekeeper, including communication services.

Overview of Container Services and Their Configuration Files

Table 19–1 gives an overview of the Services Gatekeeper configuration files.

Table 19–1 Services Gatekeeper Configuration Files

File	Contains configuration for...
Domain_home/**config/config.xml**	WebLogic Server. See "Domain Configuration files" in *Oracle WebLogic Server Understanding Domain Configuration for Oracle WebLogic Server*. **Note:** Do not edit this file manually. The following additional elements are specific to Services Gatekeeper: ▪ A <custom-resource> element, with `<name>accesstier</name>`, that specifies the location of at.xml. ▪ A <custom-resource> element, with `<name>networktier</name>`, that specifies the location of nt.xml. ▪ A set of <app-deployment> elements specific for the deployed communication services. See the relevant communication service in *Services Gatekeeper Communication Service Reference Guide*. To manage the lifecycle of communication services, use the WebLogic Server tools, such as the Administration Console's Deployment page or the command-line tools. For more information see "Overview of Common Deployment Scenarios" in *Oracle Fusion Middleware Deploying Applications to Oracle WebLogic Server.*
Domain_home/**config/custom/at.xml**	Services Gatekeeper access tier container and container services. This is a custom resource. **Note:** Do not edit this file.
Domain_home/**config/custom/nt.xml**	Services Gatekeeper network tier container and container services. This is a custom resource. **Note:** Do not edit this file.
Domain_home/**config/custom/wlng-edr.xml**	EDRs, CDRs, and alarms. This is a custom resource **Note:** Do not edit this file manually.

Finding Container services

Table 19–2 describes the Services Gatekeeper Java EE container services. The interfaces to these, with common classes, are packaged in

Middleware_home/**ocsg/server/lib/wlng/wlng.jar**

The container services are packaged in separate JARs, located in

Middleware_Home/**ocsg/server/lib/wlng/wlng.jar**

Table 19–2 Services Gatekeeper Container Services

Container service	Summary
com.bea.wlcp.wlng.core.CoreServerService	Performs initial setup during the start-up sequence.

Table 19–2 (Cont.) Services Gatekeeper Container Services

Container service	Summary
com.bea.wlcp.wlng.snmp.SNMPServerService	SNMP service.
com.bea.wlcp.wlng.event_channel.EventChannelServerService	Broadcasts arbitrary events between modules and servers in a cluster.
com.bea.wlcp.wlng.storage.StorageServerService	Storage service.
com.bea.wlcp.wlng.storage.db.DbServerService	Storage provider for persistence using direct database access.
com.bea.wlcp.wlng.storage.inval.InvalidatingServerService	Storage provider for persistence using an invalidating cache.
com.bea.wlcp.wlng.storage.configuration.ConfigurationStoreServerService	Storage provider for configuration settings.
com.bea.wlcp.wlng.edr.EdrServerService	EDR service.
com.bea.wlcp.wlng.core.budget.BudgetServerService	Budget service.
com.bea.wlcp.wlng.georedundant.GeoRedundantServerService	Geographical redundancy service.
com.bea.wlcp.wlng.account.AccountServerService	Account service.
com.bea.wlcp.wlng.statistics.StatisticsServerService	Statistics service.
com.bea.wlcp.wlng.plugin.PluginManagerServerService	Plug-in manager.
com.bea.wlcp.wlng.storage.georedundant.GeoRedundantStoreServerService	Storage provider for intercepting store operations for geographically redundant stores, forwarding changes to the geo storage server service functions and reads to the local storage provider.
com.bea.wlcp.wlng.geostorage.GeoStorageServerService	Geo storage server service that has the geo storage singleton services, manages the geo-master lifecycle, performs calls remote calls to the other site, and consistency checks between sites. Also has the geo storage Mbean.
com.bea.wlcp.wlng.tier_routing.TierRoutingManagerServerService	Tier routing manager.

Patching Container Services

You patch container services using the SmartUpgrade utility. See “SmartUpgrade Support for JDBC” in *Oracle Fusion Middleware Administering JDBC Data Sources for Oracle WebLogic Server*.

You patch container services and WebLogic Server using the upgrade process. See "Understanding the 12c Upgrade Process" in *Oracle Fusion Middleware Upgrade Planning Guide.*

20

Configuring and Managing Communication Service Traffic

This chapter describes how to configure the Oracle Communications Services Gatekeeper Plug-in Manager to route traffic. It also provides a configuration workflow.

Understanding Plug-ins and the Plug-in Manager

A plug-in is the part of a communication service that routes traffic and translates that traffic from one protocol to another. This chapter primarily explains how to use the Plug-in Manager to route traffic.

Plug-ins are modules that can be plugged into an existing deployment to extend target management or other vertical functionality. Network protocol plug-ins consist of two parts: a plug-in service and a plug-in instance. The plug-in service is the deployable and updatable unit. It does not itself process any traffic.

A plug-in service has a unique ID and a version number. It is packaged in an EAR file, with other plug-ins that share the same set of application-facing interfaces.

Plug-in instances are instantiated from a plug-in service. One plug-in service can be the base for many plug-in instances. A plug-in instance:

- Has a unique ID.
- Is versioned based on the plug-in service it is instantiated from.
- Belongs to a type.
- Exposes a set of traffic interfaces to the service communication layer. This set is a Java representation of the web services interfaces exposed by the service access layer.
- Interfaces with network nodes using one or more protocols.
- Supports a set of address schemes.
- Is configured and managed independently of other instances.
- Can be assigned a node ID to be used when setting up node SLAs.

A plug-in Manager serves as a solution for performing all plug-in deployment-related activities, through GUI as well as CLIT. The plug-in manager inspects a request and determines which interface and method the request belongs to, along with the names of any arguments. This is useful for creating a service provider group and application group SLAs.

In Services Gatekeeper, plug-in instances are created using **PluginManagerMBean**. A managed bean (MBean) is a Java bean that provides a Java Management Extensions

(JMX) interface. For more information, see the entries in the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

Understanding the Plug-in Manager Execution and Evaluation Flow

The following section describes the execution and evaluation flow.

How Plug-in Manager Evaluates Application-initiated Requests

Plug-in managers ensure that an application’s request is routed to the required network protocol plug-in instance. Service interceptors are employed to decide whether to permit, deny, or stay neutral to a particular request.

When it receives a request from an application, the plug-in manager triggers a chain of service interceptors that evaluate the following information in the request:

- Data in the request originating from the application.
- Status of the plug-in (only plug-ins in active status are considered).
- Properties of the registered plug-ins, including:
  - Plug-in type.
  - Plug-in ID.

    Refer to the *Services Gatekeeper Communication Service Reference Guide* for each communication service for the plug-in service ID and plug-in type under which individual plug-ins are registered.
  - Address plan, indicating the kind of network to which a plug-in is connected. For example, E.164 for the international public telecommunication numbering plan, or SIP.
- Usage policies based on SLA settings.
- Load balancing.
- Data from external sources: any custom data-lookups implemented as service interceptors.
- Routing logic

Service interceptors evaluate the routing logic expressed in XML. The evaluation results in the selection of a plug-in instance and the plug-in manager forwards the request to the selected plug-in.

The following interceptors use data configured using the Plug-in Manager:

- CreatePluginList
- RemoveInvalidRoute

How Plug-in Manager Evaluates Network-triggered Requests

When a request is triggered from the network, the Plug-in Manager is also part of the request flow and is responsible for invoking the chain of service interceptors.

Understanding Plug-in Routing Logic

The plug-in routing logic matches data in a request, and data associated with a request, with routing logic and results in the selection of a plug-in instance. The request is handed off to the selected plug-in instance for further processing.

The routing logic offers a fine grained way to select a network protocol plug-in instance and makes it possible to select a plug-in instance, and thereby a network node, to be targeted for individual requests, based on all data available in the request.

In combination with the plug-in instance feature make it possible to single out a certain network node based on the above operands and thereby:

- Offer different QoS levels. Example: different network nodes offers different QoS, for example latencies for message delivery.
- Ensure that requests are routed to a node that can actually handle the request. Example: one SMSC is capable of handling binary content in the form of SM logos, while another is not.
- Maximize utilization of the network nodes. Example: Two network nodes offers exactly the same functionality, but one processes the request more expensively, so the selection is done based on the charging code the application supplied.

In combination with the plug-in instance feature make it possible to single out a network protocol plug-in instance even if the different plug-in instances connect to the same network node and thereby use different configurations for the request toward the network node.

For example, a network protocol plug-in offers the possibility to configure a priority parameter when sending request to the network node, by setting the parameter differently in different plug-in instances, different priorities can be used for different service providers. In the same way credentials can be mapped so the originator of the request is mapped to the request that is made to the network node.

Defining Routing Logic

The routing logic is expressed in XML and provides:

- A set of logical operators:
  - AND
  - OR
  - NOT
- Access to a set of operands:
  - The method name
  - All parameters in the request
  - The authentication data, and the data associated with the authentication data, such as service provider ID, application ID, and application instance ID.
  - The destination address in the request
  - Tunnelled parameters provided by the application as xparams in the SOAP header.

The XML based routing logic is specified per plug-in instance and routes requests to a plug-in based on logical expressions and tests that gets evaluated. Table 20–1 describes the logical operators (XML elements).

Table 20–1 Logical operators/elements for Plug-in routing

Operator/Element	Description
and	This element contains 1 or many nested elements that in turn are evaluated. This expression will only be true if all the contained elements are true.
or	Contains 1 or many nested elements that in turn are evaluated. This expression will be true if any of the contained elements are true.
not	This element can contain only one nested element. This expression results in the inverse of the nested element.

The operands are XML elements, as described in Table 20–2.

Table 20–2 Operands/elements for Plug-in routing

Operand/Element	Description
spId	The service provider ID associated with the request is compared with the value given in this element. The result is true only if there is an exact match.
appId	The application ID associated with the request is compared with the value given in this element. The result is true only if there is an exact match.
appInstanceId	The application instance ID associated with the request is compared with the value given in this element. The result is true only if there is an exact match.
destAddress	The destination address provided in the request is compared with the value given in this element. Which parameter that is considered the destination address is plug-in specific. The format of the value is a regular expression specified as a string. The result is true only if the regular expression matches. See "Specifying Address Ranges in Routes" for examples of regular expressions. The element has the optional attribute `defaultResult`. It is a Boolean attribute that affects the result if the request does not contain any destination address. If the attribute is set to false the evaluation results in false. If set to true, the evaluation results in true. Default value is false.

method	The name of the method that the application invoked is compared with the value given in this element.
	The result is true only if there is an exact match.
	For example:
	`<method>sendSms</method>`
	`<method>makeACall</method>`

Table 20–2 (Cont.) Operands/elements for Plug-in routing

Operand/Element	Description
param	A named parameter in the request is compared with the value given in this element. There are three attributes to this element: ■ name, the parameter name. The format of the attribute is the same as used when referring to parameters in the SLAs. Mandatory attribute.Use the **getServiceInfo** method of the **PluginManagerMBean** for a list of parameter names. See "PluginManagerMBean Reference" for information on using this MBean. ■ value, the value of the parameter. If the parameter is not of type String, the Java toString() method is used to convert it to a String. Mandatory attribute. ■ defaultResult, a Boolean attribute that affects the result if the request does not contain the specified name attribute. If set to false the evaluation results in false if the name attribute is not present in the request. If set to true, the evaluation results in true if the name attribute is not present in the request. Default value is false. Optional attribute. The result is true only if there is a match or if defaultResult is set to true and the name attribute is not present at all in the request. The result is true only if there is a match. The value can be expressed as a regular expression. For example: `<param name="arg0.senderName" value="tel:123456"/>` <param name="arg0.senderName" value="tel:123.*"/>

xparam	A parameter supplied in the request as a tunnelled parameter (xparam) in the request is compared with the value given in this element. There are three attributes to this element:
	■ name, corresponds to the contents of the key attribute in the param element in the request. Mandatory attribute.
	■ value, corresponds to the contents of the value attribute in the param element in the request.Mandatory attribute.
	■ defaultResult, a Boolean attribute that affects the result if the request does not contain the specified name attribute. If set to false the evaluation results in false if the name attribute is not present in the request. If set to true, the evaluation results in true if the if the name attribute is not present in the request. Default value is false. Optional attribute.
	The result is true only if there is a match or if defaultResult is set to true and the name attribute is not present at all in the request.
	The result is true only if there is a match. The value can be expressed as a regular expression.
	For example, the tunneled parameter is defined as:
	`<xparams>`
	`  <param key="aParameterName" value="aParameterValue"`
	`/>`
	`</xparams>`
	A match occurs if the xparam element in the routing configuration is
	`<xparam name="aParameterName" value="aParameterValue"/>`

The plug-in routing configuration XML document is validated when it is provisioned. The XML is validated according to the XSD. See "Plug-in Routing XSD".

For a set of examples, see "Plug-in Routing Configuration Examples".

Plug-in Routing Configuration Examples

The example in Example 20–1 matches requests sent to an address starting with `tel:123`. If the request does not contain an address, this route does not match.

Example 20–1 Plug-in Routing Configuration Example 1

```
<?xml version="1.0" encoding="UTF-8"?>
<route xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="route.xsd">
  <destAddress>tel:123.*</destAddress>
</route>
```

The example in Example 20–2 matches any address and also matches any request that does not contain an address.

Example 20–2 Plug-in Routing Configuration Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<route xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="route.xsd">
  <destAddress defaultResult="true">.*</destAddress>
</route>
```

The example in Example 20–3 matches requests sent from the service provider with the ID `sp1`, and the application with the ID `app1`.

Example 20–3 Plug-in Routing Configuration Example 3

```
<?xml version="1.0" encoding="UTF-8"?>
<route xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="route.xsd">
  <and>
    <spId>sp1</appId>
    <appId>app1</appId>
  </and>
</route>
```

The example in Example 20–4 matches all requests except the ones where the operation is `sendSmsRingtone` and either sent from and application using application instance ID `appInst1` or the operation is `sendSms` withe the value of the parameter `senderName` is `tel:123456`.

Example 20–4 Plug-in Routing Configuration Example 4

```
<?xml version="1.0" encoding="UTF-8"?>
<route xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="route.xsd">
  <and>
    <not><method>sendSmsRingtone</method></not>
    <or>
      <appInstanceId>appInst1</appInstanceId>
      <and>
        <method>sendSms</method>
        <param name="arg0.senderName" value="tel:123456"/>
      </and>
    </or>
  </and>
</route>
```

Plug-in Routing XSD

Following is the XSD to use when creating a plug-in routing configuration XML file.

Example 20–5 route.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="route">
    <xs:complexType>
      <xs:group ref="ConditionGroup" minOccurs="1" maxOccurs="1"/>
    </xs:complexType>
  </xs:element>

  <xs:group name="ConditionGroup">
    <xs:choice>
      <xs:element name="and" type="AndType"/>
      <xs:element name="or" type="OrType"/>
      <xs:element name="not" type="NotType"/>
      <xs:element name="spId" type="xs:string"/>
      <xs:element name="appId" type="xs:string"/>
      <xs:element name="appInstanceId" type="xs:string"/>
      <xs:element name="destAddress" type="AddressType"/>
      <xs:element name="method" type="xs:string"/>
      <xs:element name="param" type="ParamType"/>
      <xs:element name="xparam" type="ParamType"/>
    </xs:choice>
  </xs:group>

  <xs:complexType name="AndType">
    <xs:group ref="ConditionGroup" minOccurs="1" maxOccurs="unbounded"/>
  </xs:complexType>

  <xs:complexType name="OrType">
    <xs:group ref="ConditionGroup" minOccurs="1" maxOccurs="unbounded"/>
  </xs:complexType>

  <xs:complexType name="NotType">
    <xs:group ref="ConditionGroup" minOccurs="1" maxOccurs="1"/>
  </xs:complexType>

  <xs:complexType name="AddressType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="defaultResult" type="xs:boolean" default="false"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="ParamType">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="value" type="xs:string" use="required"/>
    <xs:attribute name="defaultResult" type="xs:boolean" default="false"/>
  </xs:complexType>
</xs:schema>
```

Specifying Address Ranges in Routes

Address ranges are specified using UNIX regular expressions. A few examples are given below:

- ^.* specifies a route that matches all addresses.
- ^[0-5].* specifies a route that matches all address strings starting with 0, 1, 2, 3, 4, or 5.
- ^.*[6-9]$ specifies a route that matches all address strings ending with 6, 7, 8, or 9.
- ^46.* specifies a route that matches all address string starting with 46.
- ^46.{8}$ specifies a route that matches all address strings starting with 46 that contain exactly 10 total digits.
- ^.*@.*\.com$ specifies a route matching all mail addresses in the com domain. The dot in .com must be written "\.".

In the examples:

- ^ indicates the beginning of the string.
- . matches anything except a new-line. That is, "a.b" matches any three-character string that begins with a and ends with b.
- * is a suffix that means the preceding regular expression is to be repeated zero or more times. That is, the "." in the expression “^46.*" is repeated until the whole string is matched.
- $ is an indicator of end of line (or end of string).

The address scheme can be included in the expression. If not specified, any scheme will match. Examples:

- `tel:^46.*` matches all tel (E.164) numbers starting with 46.
- `sip:.*` matches any SIP address.

Understanding Plug-in Routes and Routing Logic

Plug-in routing logic is an extension of the plug-in routes and these **PluginManagerMBean** methods:

- **setRouteConfig**
- **getRouteConfig**

Plug-in routes added using the **addRoute** method, are converted into plug-in routing logic where a route is added as a logical OR given that the route that is being modified contains `<or>` and `<destAddress defaultResult="true">` elements only.

See "PluginManagerMBean Reference" for information on using this MBean.

Configuring Plug-in Manager

Configuring the Plug-in Manager tasks include:

- Configuring the general behavior of the plug-in manager.
- Administering routes, which is tightly coupled to configuring individual communication services.

Following is an outline for configuring the Plug-in Manager using the **PluginManagerMBean**:

1. Specify whether or not to use policy-based routing in the **PolicyBasedRouting**. field. Policy based routing is necessary in order to enforce node SLAs.
2. Specify the network protocol plug-in behavior when it is being deployed or re-deployed in the **ForceConnectInResuming** field.

See "PluginManagerMBean Reference" for information on using this MBean..

Creating a Plug-in instance

Following is an outline for creating an instance of a plug-in service using the **PluginManagerMBean**:

1. Find the plug-in service ID for the service you want to use to create the plug-in instance. The plug-in service IDs are listed under the heading “Properties” in the sections describing the management of each plug-in. To get a list of plug-in service IDs, use the **listPluginServices** method.
2. Use the **createPluginInstance** method to create the instance. The Plug-in Instance ID supplied as a parameter will be used to identify the instance for all routing and administration.
3. To destroy an instance, use the **destroyPluginInstance** method.

See "PluginManagerMBean Reference" for information on using this MBean.

Administering Plug-in Routing Logic and Node IDs

You administer routes using these **PluginManagerMBean** operations:

- To add new routing logic: **getRouteConfig**
- To change or remove routing logic: **getRouteConfig** and **setRouteConfig**
- To list all existing routes: **listRoutes**
- To get information about a specific plug-in instance: **getPluginServiceInfo**
- To get a list of all registered plug-in instances: **listPluginInstances**
- To get a list of all registered plug-in services: **listPluginServices**
- To define the node ID: **setPluginNodeId**.

  A node ID is assigned to one or more Plug-in Instance IDs to enforce node SLAs. The node ID is then referred to in the node SLAs.

See "PluginManagerMBean Reference" for information on this MBean.

Adding Bulk Messaging Support to a Communication Service

If a communication service does not already support bulk messaging, you can add support for it. If you want bulk messaging applied to the message address, have the **PluginFactory** implement the **BulkRequestFactory**, and implement the **createSingleAddressRequest** method.

To make bulk messaging apply to other types of information in the request, use these general steps:

1. Create a new class that extends **AddressRequestInfo** to contain the information after the message is split.

2. Create a new xparameter or other identifier in the original bulk request.
3. Define a new schema for the information in the bulk request, and add a process for it in the **createRequestInfo** method in the **PluginFactory**.
4. Set the information back in the **createSingleAddressRequest** method in the **PluginFactory**.

PluginManagerMBean Reference

Set field values and use methods from the Administration Console by selecting **Container**, then **Services** followed by **PluginManager**. Alternately, use a Java application. For information on the methods and fields of the supported MBeans, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

21

Upgrading and Redeploying Communication Services and Service Interceptors

This chapter explains how to upgrade and redeploy the Oracle Communications Services Gatekeeper communication services and service interceptors.

Understanding the Production Redeployment Process

Services Gatekeeper supports seamless upgrades of communication services and service interceptors without service interruption (production redeployment). You can upgrade these services without disrupting the traffic flow and without having to restart servers. This preserves the full processing capacity of your Services Gatekeeper implementation during the hitless upgrade.

For instructions on how to perform a hitless upgrade see "Client Considerations When Redeploying a Web Service" in *Oracle Fusion Middleware Getting started with JAX-WS Web Services for Oracle WebLogic Server Guide*.

You use production redeployment to upgrade Services Gatekeeper container services. See “Deploying Applications in a Production Environment” in *Oracle WebLogic Server Upgrade Guide for Oracle WebLogic Server*.

Performing a Hitless Upgrade

Services Gatekeeper uses the WebLogic Server mechanism for production redeployment.

Following is the syntax for using **weblogic.Deployer** to do an upgrade:

```
java weblogic.Deployer -adminurl <adminhost>:adminport -user admin user -password
password -graceful -rmiGracePeriod grace period -redeploy -name name -source EAR
file -targets Cluster name
```

In addition to the **-graceful** and **-rmiGracePeriod** flags, the **-retiretimeout time period** flag can be used to stop the application after the given time period.

In the code snippet below, the network tier part of the Parlay X 2.1 Multimedia Messaging communication service on a cluster named **WLNG_NT_Cluster** is redeployed:

```
java weblogic.Deployer -adminurl <adminhost>:7001 -user weblogic -password
weblogic -redeploy -rmiGracePeriod 30 -name wlng_nt_multimedia_messaging_px21
-source ./wlng_nt_multimedia_messaging_px21.ear -targets WLNG_NT_Cluster
```

The EARs are version-controlled, so the **meta-inf\MANIFEST.MF** file in the EAR must be updated with which version the EAR has:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.6.5
Created-By: R27.6.0-101-94136-1.6.0_14-20080116-2103-windows-ia32 (BEA
  Systems, Inc.)
Bundle-Name: wlng_at_multimedia_messaging_px21
Bundle-Version: $<version>
Bundle-Vendor: BEA Systems, Inc
Weblogic-Application-Version: $<version>
```

Performing Production Redeployment

Production redeployment involves deploying a new version of an application alongside an older version of the same application. WebLogic Server automatically manages client connections so that only new client requests are directed to the new version. Clients already connected to the application during the redeployment continue to use the older version of the application until it has processed any pending request, at which point WebLogic Server automatically retires the older application. In this context, applications are network protocol plug-ins.

To support the production redeployment strategy, a plug-in must support *graceful shutdown*. It should track any pending requests and ensures that it does not shut down with any pending unprocessed requests. The WebLogic Server container takes care of parts of this, but all plug-ins need to perform protocol-specific cleanup of any connections used for network traffic and assert that all traffic is diverted to a *new version* of the module.

Production redeployments are performed on EAR file level, which means that all network protocol plug-ins that are tied to an application-facing interface are updated.

Service interceptors support production redeployment.

Understanding the Redeployment Sequence

A production redeployment uses this sequence:

1. The redeployment operation is triggered. It is performed on the administration server and targeted to all servers in the cluster.
2. The old version of the plug-in stops accepting new requests at the same time that the new version of the plug-in starts accepting them. At this point, both plug-ins are operational. The old version does not accept new requests; it processes only pending requests.
3. When the old version has no pending requests, it is undeployed automatically.

While both the new version and the old version of the plug-in are active, both plug-ins are registered in the Plug-in Manager, with the version of the plug-in indicated. See "Configuring and Managing Communication Service Traffic" for details.

Understanding Redeployment Requirements

For production redeployment to work:

- The old version of the plug-in must:
  - Perform protocol-specific cleanup of any connections or sessions toward the network nodes.
  - Assert that all traffic is diverted to the new version of the module.

  - Track pending requests, if the network protocol used by the plug-in is not based on HTTP or RMI. Pending requests over HTTP and RMI are tracked by WebLogic Server mechanisms.

    > **Caution:** If this is not done properly, the old version could be undeployed prematurely.

  - Report back to the container when ready to be undeployed.
- The new version of the plug-in must not:
  - Change the schema used to define the data model for the named store in the StorageService.
  - Change the interface used between the access tier and the network tier in the communication service.

Typical Production Redeployment Scenarios

The following sections explain redplyoment scenarios for different Services Gatekeeper components.

The EAR names for each communication service and the JAR names for the specific network protocol plug-ins contained within it can be found under the heading "Properties for" the communication service in *Services Gatekeeper Communication Service Reference Guide*. See "Deploying and Administering Communication Services" for a description of how individual communication services are packaged.

Redeploying Communication Services Using HTTP-based Network Protocol Plug-ins

HTTP-based network protocol plug-ins do not establish HTTP sessions when sending requests to or receiving requests from to the underlying network nodes. In the absence of sessions, the WebLogic Server servlet container chooses the most recent/active version of a particular WAR module whenever new network triggered requests arrive. In-progress network-triggered request are allowed to complete through the shutdown of EAR-scoped work managers. Before the older version of the EAR is retired, all of the pending work related to handling of the network-triggered traffic is completed.

Redeploying the Parlay X 2.1 Short Messaging-Binary SMA/SMPP Communication Service

The Parlay X 2.1 Short Messaging-Binary SMS/SMPP plug-in handles graceful transition to ADMIN state as part of the server life cycle or during EAR upgrade. Draining both application-initiated and network-triggered traffic out of the older version is handled internally by the plug-in. When a new version of the plug-in becomes available, the older version begins the process of completing the processing of all application-initiated traffic it is responsible for. When all of the application-initiated traffic has switched to the newer version of the plug-in, the suspension of the network-triggered traffic in the older version is initiated. During the suspension of network triggered traffic, all new requests from the Short Message Service Center (SMSC) are rejected with `ESME_RSYSERR` error code in the responses. The in-progress network-triggered requests are acknowledged to the SMSC as they are processed by the application or are stored for later retrieval/processing. For requests that have received the `ESME_RSYSERR` response, the SMSC is expected to failover to the newer plug-in version. Application-initiated traffic is handled first in order to minimize the window during which the `ESME_RSYSERR` command status error codes are sent to the SMSC.

The new version of the plug-in will bind new receiver connections before the old version disconnects its receiver connections. This will ensure continuous network triggered traffic. No state is associated with the bind itself and any plug-in on any server is able to handle any network triggered traffic regardless of what server created the notifications. Only the volatile conversational state needs to be handled before the plug-in unbinds any of the ongoing connections toward the SMSC.

Hitless upgrade of both Transmitter/Receiver and Transceiver bind types are supported.

Redeploying Communication Services With OSA/Parlay Type Plug-ins

The OSA access functionality is embedded in the network tier communication service EAR file, and any upgrades or updates to this part will follow the new EAR.

All application-initiated traffic is switched to the newer version of a plug-in as soon as it becomes available. This prevents newer call sessions from being started against the older version. For application-initiated traffic, the new OSA/Parlay plug-in might get traffic that concerns sessions started with the previous version in the case of long-lived sessions, such as call control. In this case, the old version must still be able to handle all callbacks from the OSA/Parlay Gateway that might be related to the traffic handled by the new version.

The OSA/Parlay plug-in counts the number of unfinished active sessions handled when it is being retired, and uses a barrier to inform the WebLogic Server container when all sessions have finished. This is the case for the Parlay X 2.1 Third Party Call/MPCC, Call Notification/MPCC, and Audio Call/MPCC-CUI plug-ins, where there are long lived sessions.

Each plug-in also has a time-out to provide a reasonable boundary for the overall graceful retirement duration.

The OSA/Parlay plug-ins use the managers replicated by the OSA Access module to create new notifications and set new callbacks.

Redeploying Communication Services With SIP Type Plug-ins

SIP-based Communication Services consist of an Services Gatekeeper plug-ins and a SIP application. The Services Gatekeeper side plug-ins can be hitlessly upgraded the same way as other Services Gatekeeper plug-ins.

Graceful retirement is not supported for SIP applications. For the SIP servlet part of the plug-ins, Services Gatekeeper supports the same `-retiretimeout` option as WebLogic SIP Server.

Redeploying Extended Web Service Subscriber Profile/LSAP Communication Services

The Subscriber Profile/LDAP plug-in uses synchronous application-initiated traffic only. In-flight traffic is tracked and use a barrier is used to notify the WebLogic Server container when all traffic is completed.

The Subscriber Profile/LDAP plug-in uses synchronous application-initiated traffic only. In-flight traffic is tracked and uses a barrier to notify the WebLogic Server container when all traffic is completed.

Redeploying Native SMPP Communication Services

Native SMPP communication service does not support hitless upgrades. The communication service needs to be undeployed and deployed to be updated.

Native MM7 communication service handles hitless upgrades as described for HTTP-based network protocol plug-ins.

22

Managing and Configuring the Tier Routing Manager

This chapter describes how to configure and maintain the Oracle Communications Services Gatekeeper Tier Routing Manager. It also provides a workflow for the configuration.

Understanding the Tier Routing Manager

The Tier Routing Manager routes network-triggered communication requests from Services Gatekeeper communication services to the appropriate exposes service and access tier cluster.

Applications interact with these Service Facades when using Services Gatekeeper communication services:

- SOAP
- SOA
- RESTful
- Native

Each Service Facades uses a Service Enabler to connect to the telecom network.

Application-initiated requests are automatically routed from a Service Facade to the proper Service Enabler, since there is a many-to-one (n:1) relationship between Service Facades and Service Enablers.

Network-triggered requests are routed to the correct Service Facade by an identifier which specifies the Service Facade used when setting up the subscription. The Tier Routing Manager uses this identifier to route network-triggered requests to the appropriate Service Facade, or more specifically to the correct access tier cluster. There is one access tier cluster with SOA Service Facades, one cluster with RESTful Service Facades, and one cluster with SOAP Service Facades.

The routing uses *tier routes* to identify to which cluster to route the request. A tier route is identified by an index and has the following properties:

- An endpoint expression
- The name of the cluster for which the route is valid

When an application wishes to receive traffic from the network, it subscribes to notifications for network triggered events. As a part of the subscription, the application provides a URL for the endpoint to which notifications should be sent.

The SOA Service Facades uses the application-provided endpoint URL and rewrites it, embedding information that it was the SOA Service Facade that was used to set up the notification along with the original URL. This new URL is passed used the request to the Service Enabler. The SOA Service Facade replaces the original callback URL with a new URL that points to the Oracle Service Bus Proxy Service and adds the parameter **realUrl** that has the value of the original callback URL.

For example, if the original callback URL is:

```
http://somehost/services/SmsNotification
```

the rewritten callback URL is

```
http://soahost/proxySms/SmsNotification?realUrl=http://somehost/services/SmsNotifi
cation
```

The RESTful Service Facades always gets a callback URL starting with /bayeux/ from the application, so the URL itself provides information that the subscription originated from the RESTful Service Facade.

When a network-triggered request arrives at the Tier Routing Manager, it inspects the endpoint (rewritten) URL to extract from it the Service Facade to which it should be routed. It looks at the pattern of the URL and matches it to the configured endpoint expression type. When a match is found, it resolves the cluster name and passes the request on to the appropriate cluster.

> **Note:** Some network protocol plug-ins support off-line provisioning for subscriptions for notifications. The callback URL used for these subscriptions must be formatted according to the URL rewrite pattern of the appropriate Service Facade.

See Table 24–1 for examples of endpoint expressions.

Table 22–1 Examples of Tier Routes

Endpoint expressions	Description
.*realUrl=.*	Routes to a SOA Service Facade cluster, since the parameter realURL is present in the callback endpoint.
/bayeux.*	Routes to a RESTful Service Facade cluster, since the parameter bayeux is present in the callback endpoint.
http://.*mydomain.com/*	Routes to a specific cluster when the callback endpoint is within the domain mydomain.com. This makes it possible to inspect in which domain the callback endpoint is, and use one access cluster for a given set of service providers and another for another set of service providers. It makes it possible to use one Access tier cluster for applications residing in the network operator's intranet and another for applications hosted by service providers outside the network operator's domain.

Configuring Tier Routing

You administer routes with these methods to **TierRoutingManagerMBean**:

- To add a new tier route, use **addTierRoute**
- To remove a tier route, use **removeTierRoute**

- To list all existing tier routes, use **listTierRoutes**

Set field values and use methods from the Administration Console **Container Services** and then **TierRoutingManager**. Alternately, use a Java application. For information on the methods and fields, see the "All Classes" section of *Services Gatekeeper OAM Java API Reference*.

23

Charging and Integrating Billing

This chapter explains the Oracle Communications Services Gatekeeper charging functionality and provides an overview of how you can integrate an external billing system.

About Charging

You can select the type of charging to each application service. An application can use one or both of the following types of charging:

- Content-based charging, which is charging based on the content or value of the service used
- CDR-based charging, which is charging based on the period of time a service is used (the duration) or the time of day that a service is used

Content-Based Charging

Content or value based charging is used to charge a user based on the variable value of a used service rather than on the duration of the use on flat rates. This can be used, for example, when downloading music video clips or in mobile commerce applications. Services Gatekeeper supports both prepaid and postpaid user accounts.

CDR-Based Charging

Call Data Records (CDRs) are used for charging based either on the duration of use or on access to certain per-use services. Charging for calls is typically based on duration. Per-use might be used, for example, to charge for locating a terminal.

CDR data can be stored in the Services Gatekeeper internal charging database or retrieved in real time by billing and post processing systems through a billing gateway. Retrieving data in real time requires integrating with the billing gateway. See "Billing System Integration" for more information.

Charging data is generated every time an application uses a communication service. The charging data is recorded by the communication service during the period that the application interacts with the network. When the interaction is closed, the communication service stores the charging data as a CDR in the Services Gatekeeper database. If Services Gatekeeper is integrated with a billing gateway, the charging data is sent directly to the billing gateway.

Diameter Charging Support

Services Gatekeeper has specific support for using the Diameter protocol for online (Ro) and offline (Rf) charging. The Parlay X 3.0 Payment communications services can be used for online charging using Diameter. Offline charging can be integrated using the CDR to Diameter module, which converts CDRs generated by Services Gatekeeper into the appropriate format for Diameter charging requests and then sends them on to an offline Diameter server.

Billing System Integration

You can integrate Services Gatekeeper with external billing systems. The billing system can either receive charging data directly or automatically retrieve information from the Services Gatekeeper database. CDRs can be customized to fit the format and behavior requirements of these billing systems. Services Gatekeeper supports integration with the Oracle Communications Elastic Charging Engine and Oracle Communications Billing and Revenue Management products, but can be used with any billing system that supports the Diameter protocol. Billing systems that support offline charging (Diameter Rf) use the Services Gatekeeper CDR module. Billing systems that support online charging (Diameter Ro) use either the Services Gatekeeper credit control interceptors for in-traffic credit checks or the Parlay X 3.0 Payment/Diameter communication service for direct billing.

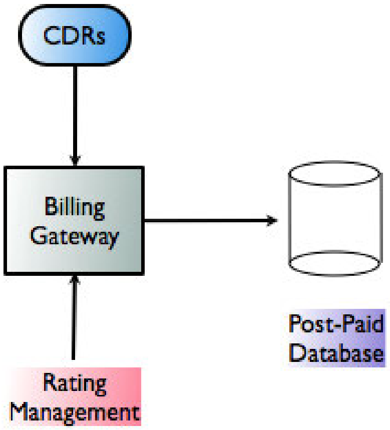
See "Implementing Diameter Ro Online Charging" or "Implementing Diameter Rf Offline Charging" for information on how to integrate Services Gatekeeper with an online or offline billing system. See "*Parlay X 3.0 Payment/Diameter*" in *Services Gatekeeper Communication Service Reference Guide* for details on integrating Services Gatekeeper with direct billing.

Billing Gateways

Real-time settlement of prepaid accounts using CDR-based charging requires integration through a billing gateway. A billing gateway can also be used to support charging of postpaid services.

When integrating through a billing gateway, the billing gateway retrieves the CDRs in real time through an external JMS-based charging listener. Rating, rating management, billing information storage, and prepaid accounts settlement are handled by the billing gateway. The workflow is shown in Figure 23–1.

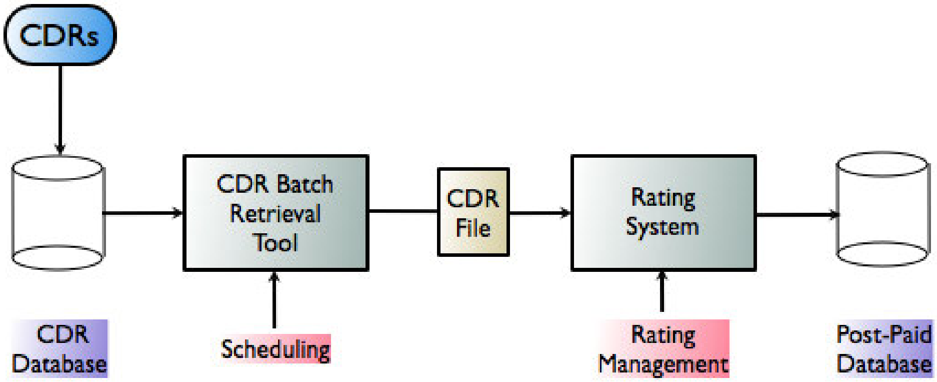
Figure 23–1 Billing Integration Through a Billing Gateway



CDR Database

If an application uses postpaid accounts, you can integrate billing by retrieving CDRs that have been stored in the Services Gatekeeper database. When you integrate billing using this method, a CDR batch retrieval tool retrieves the CDRs from the database and stores them in a file. The CDR file is processed by a rating system that transforms it into billing information and then stores it in a postpaid accounts database. The workflow is shown in Figure 23–2.

Figure 23–2 Billing Integration Using a CDR Database



24

Implementing Diameter Ro Online Charging

This chapter describes how to manage and configure the Oracle Communications Services Gatekeeper Credit Control Interceptors to work with Diameter online (Ro) credit checks.

Understanding Credit Control Interceptors

The Credit Control Interceptors perform credit checks toward a Diameter server using the Diameter Ro interface. This allows for easy integration with charging systems that support Diameter Ro online charging. Oracle Communications Elastic Charging Engine or Oracle Communications Billing and Revenue Management, can be used out-of-the-box with Services Gatekeeper.

Traffic flowing through any Communication Service is intercepted and the data in the request, with other configuration data, is passed on to a Diameter server for credit control verification. Both application-initiated and network-triggered requests are intercepted, there is one interceptor for each direction.

The credit control check is performed either synchronous or asynchronous.

Using Application-Initiated Requests

In the synchronous case, money is reserved before the request is passed on to the telecom network node. When the request has been successfully handed off and the thread of execution is returning, the reservation is committed. In case the requests fails, the reservation is canceled.

In the asynchronous case, money is reserved before the request is passed on to the telecom network node. When a correlated request arrives to the interceptors, the reservation is committed. In case the correlated request does not arrive within a set period, the reservation is canceled.

Using Network-Triggered Requests

Money is reserved before the request is passed on to the application. When the request has been successfully handed off and the thread of execution is returning, the reservation is committed. In case the requests fails, the reservation is canceled.

Only synchronous reservations are supported.

Using Credit Control Interception Points

Only requests that are explicitly defined to be subject to credit control checks are passed on to the Diameter server.

You can map a parameter in the request to be passed in the Diameter request. Or define a static value for the following attribute value pairs:

- User-Name
- OCSG-Charge-Description
- Service-Context-Id
- Unit-Value.Exponent and Unit-Value.Value-Digits
- Currency-Code

The mapping is expressed in XML and provisioned as a service level agreement on service provider group or application group level.

Writing Credit Control SLAs

Credit Control SLAs define which methods are subject to credit control, how the credit control is performed, and which data to provide to the Diameter server. Table 24–1 describes the structure and content of a credit control SLA.

Table 24–1 Structure and contents of a credit control SLA

Tag	Description
<CCInterceptions>	Main tag. Defines a set of interception points for credit controls. Contains: <CCInterception>, one (1) or more
<CCInterception>	Defines how and when to trigger a credit control reservation. Used both in asynchronous and synchronous credit control checks. This element has the following attributes: ■ InterfaceName, which defines the Service Gatekeeper interface that the method defined in the attribute methodName belongs to. ■ methodName, which defines the name of the method that the credit control check applies to. See "Configuring and Managing Communication Service Traffic" for information on how to get a list of interface names and methods. For example, the interface name for Parlay X 2.1 Short Messaging and RESTful Short Messaging send SMS is **com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin** and the method that sends the SMS is **sendSms**. Contains: <SubscriptionId>, exactly one (1). <OCSGChargeDescription>, zero (0) or one (1). <ServiceContextId>, zero (0) or one (1). <Amount>, zero (0) or one (1). <Currency>, zero (0) or one (1). <AsynchronousCommit>, zero (0) or one (1).

Table 24–1 (Cont.) Structure and contents of a credit control SLA

Tag	Description
<SubscriptionId>	Parent tag: <CCInterception>. Defines how the Diameter AVP User-Name shall be populated. The alternatives are to dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the AVP shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword **Static** to define a static value. ■ value, that defines what to send in the AVP. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use. If the attribute type is set to **Static**, enter the string to use in the AVP. See "Defining Static and Dynamic Parameter Mappings"for more information.
<OCSGChargeDescription>	Parent tag: <CCInterception>. Defines how the Diameter AVP OCSG-Charge-Description shall be populated. The alternatives are to either dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the AVP shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword **Static** to define a static value. ■ value, that defines what to send in the AVP. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use, If the attribute type is set to **Static**, enter the string to use in the AVP. See "Defining Static and Dynamic Parameter Mappings" for more information.
<ServiceContextId >	Parent tag: <CCInterception>. Defines how the Diameter AVP Service-Context-Id shall be populated. The alternatives are to either dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the AVP shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword Static to define a static value. ■ value, that defines what to send in the AVP. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use. If the attribute type is set to **Static**, enter the string to use in the AVP. See "Defining Static and Dynamic Parameter Mappings" for more information. If not present, the value is set to current time, given in milliseconds, from 1970.

Table 24–1 (Cont.) Structure and contents of a credit control SLA

Tag	Description
<OCSGChargeDescription >	Parent tag: <CCInterception>. Defines how the Diameter AVP OCSG-Charge-Description shall be populated. The alternatives are to either dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the AVP shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword **Static** to defined a static value. ■ value, that defines what to send in the AVP. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use. If the attribute type is set to **Static**, enter the string to use in the AVP. See "Defining Static and Dynamic Parameter Mappings" for more information.
<Amount>	Parent tag: <CCInterception>. Defines how the Diameter AVPs Unit-Value.Exponent and Unit-Value.Value-Digits shall be populated. The alternatives are to either dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the AVP shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword **Static** to defined a static value. ■ value, that defines what to send in the AVP. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use. If the attribute type is set to **Static**, enter the string to use in the AVP. See "Defining Static and Dynamic Parameter Mappings" for more information.
<Currency>	Parent tag: <CCInterception>. Defines how the Diameter AVP Currency-Code shall be populated. The alternatives are to dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the AVP shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword **Static** to defined a static value. ■ value, that defines what to send in the AVP. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use. If the attribute type is set to **Static**, enter the string to use in the AVP. See "Defining Static and Dynamic Parameter Mappings" for more information.

Table 24–1 (Cont.) Structure and contents of a credit control SLA

Tag	Description
<AsynchronousCommit>	Parent tag: <CCInterception> Defines how and when to trigger charging of a previously reserved amount and which data to use. If present, the charging is asynchronous. If not present, the charging is synchronous. Has the following attributes: ■ InterfaceName, which defines the Service Gatekeeper interface that the method defined in the attribute methodName belongs to. ■ methodName, which defines the name of the method that the credit control check applies to. See "Configuring and Managing Communication Service Traffic" for information on how to get a list of interface names and methods. For example, the interface name for Parlay X 2.1 Short Messaging and RESTful Short Messaging call-back interface is **com.bea.wlcp.wlng.px21.callback.SmsNotificationCallback** and the method that contains a delivery notification is **notifySmsReception**. Contains: ■ <ReservationCorrelator>, zero (0) or one (1). ■ <CommitCorrelator>, exactly one (1). See "Correlating Reservation and Commit Triggers in Asynchronous Credit Control Checks" for more information.
<ReservationCorrelator>	Parent tag: <AsynchronousCommit> Defines what to use as an ID, or correlator, for the reservation part of an asynchronous credit control check. The alternatives are to dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the correlator shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword **Static** to define a static value. ■ value, that defines what to use as a correlator. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use. If the attribute type is set to **Static**, enter the string to use. See "Defining Static and Dynamic Parameter Mappings" for more information. **Note:** The parameter to use is any of the parameters in the method of the reservation request as defined in the <CCInterception> tag. Normally, the correlator used to correlate asynchronous request-response pairs is used.

Table 24–1 (Cont.) Structure and contents of a credit control SLA

Tag	Description
<CommitCorrelator>	Parent tag: <AsynchronousCommit> Defines what to use as an ID, or correlator, for the commit part of an asynchronous credit control check. The alternatives are to dynamically fetch the value from the request that is being subject to credit control or to pass in a configurable value. This element has the following attributes: ■ type, that defines if the value of the correlator shall be fetched dynamically or statically. Use the keyword **Dynamic** to fetch the value from the request. Use the keyword **Static** to define a static value. ■ value, that defines what to use as a correlator. If the attribute type is set to **Dynamic**, use the fully qualified name of the parameter to use. If the attribute type is set to **Static**, enter the string to use. **Note:** The parameter to use is the parameter of the reservation request as defined in the <CCInterception> tag. Normally, the correlator used to correlate asynchronous request-response pairs is used.

Defining Static and Dynamic Parameter Mappings

The Credit Control Interceptors can map parameters in two ways:

- Static
- Dynamic

Mappings are defined using two components: type and value. The mappings are expressed as attributes to a subset of the Credit Control SLA elements, see Table 24–1.

When type is set to **Static**, the mapping is static, which means that the attribute value is set to, and treated as an arbitrary string

When type is set to **Dynamic**, the mapping is dynamic, which means that the attribute value is set to the content of a parameter in the request that is subject to credit control checks. Which parameter to chose is configurable by referring to the Java representation of the parameter defined in the WSDL. See "Configuring and Managing Communication Service Traffic" for information on how to get a list of valid parameter names. The representation is `arg0.`<*name of parameter as defined in WSDL*>.

Depending on which element the parameter mapping exists, the value is passed on to a DIAMTER AVP or used as a key for reservations.

Here is an example of a static parameter mapping:

```
<OCSGChargeDescription type="static" value ="Static value for reservation."/>
```

Here is an example of a dynamic parameter mapping:

```
<Amount type="dynamic" value ="arg0.charging.amount"/>
```

Correlating Reservation and Commit Triggers in Asynchronous Credit Control Checks

In both synchronous and asynchronous credit control checks, the request that triggers a reservation is defined. This is done using the `<CCInterception>` tag.

For asynchronous credit control checks, the ID, or correlator, ties together the reservation request and the commit request. This is done in the same manner as parameter mappings are defined; see "Defining Static and Dynamic Parameter Mappings" for more information. There is one parameter mapping for the reservation, and one for the commit.

In most use cases the correlators are fetched dynamically from the requests. The reservation correlator is fetched from a parameter in the method in the interface defined for the trigger. The commit correlator is fetched from the request that commits the reservation. Since the commit correlator is a parameter in the method that triggers the commit the interface, method, and parameter must be defined explicitly since it is different from the method that triggers the reservations.

For each request, the two correlators are compared and in the case of a match, the reservation is committed. Reservations time-out after a certain time period and the reservation is released. The time-out is defined in the store configuration for the interceptor.

Example Credit Control SLA

Example 24–1 shows a Credit Control SLA with two different credit control checks.

The firsts check is done asynchronously. The reservation is done when the application sends a **sendSms** request. In this case, the charging description is a static String, while all other are picked up dynamically from the request. The reservation is committed when a delivery notification is sent to the application using the method **notifySmsReception** and the parameter correlator in the **sendSms** request is identical with the parameter correlator in the **notifySmsReception** request.

The second check is done synchronously. The reservation is done when the application sends a **getLocation** request. The reservation is committed when the request has successfully returned from the network node.

Example 24–1 Example Credit Control SLA

```
<?xml version="1.0" encoding="UTF-8"?>
<CCInterceptions>
<CCInterception
interfaceName="com.bea.wlcp.wlng.px21.plugin.SendSmsPlugin"
methodName="sendSms">
<SubscriptionId type="dynamic" value ="arg0.senderName"/>
<OCSGChargeDescription
type="static" value ="Static description for reservation."/>
<ServiceContextId type="dynamic" value ="arg0.charging.code"/>
<Amount type="dynamic" value ="arg0.charging.amount"/>
<Currency type="dynamic" value ="arg0.charging.currency"/>
<AsynchronousCommit
interfaceName="com.bea.wlcp.wlng.px21.callback.SmsNotificationCallback"
methodName="notifySmsReception">
<ReservationCorrelator
type="dynamic" value="arg0.receiptRequest.correlator"/>
<CommitCorrelator type="dynamic" value="arg1.correlator"/>
</AsynchronousCommit>
</CCInterception>
<CCInterception
interfaceName="com.bea.wlcp.wlng.px21.plugin.TerminalLocationPlugin"
methodName="getLocation">
<SubscriptionId type="dynamic" value="arg0.address"/>
<OCSGChargeDescription
type="static" value ="Static dummy description."/>
```

```
<ServiceContextId type="static" value="Static dummy code."/>
<Amount type="static" value="2"/>
<Currency type="dynamic" value="USD"/>
</CCInterception>
</CCInterceptions>
```

Configuring, Managing, and Provisioning Credit Control Interceptors

The Credit Control interceptors are deployed as regular EARs, but they are not registered with the **InterceptorManager** until a connection is established with the Diameter server. They are de-registered when the connection to the server is connection is closed.

The interceptors are configured and managed using an MBean. For more information about service interceptors, see the discussion on service intercepters in *Services Gatekeeper Extension Developer's Guide*.

The Credit Control SLAs are provisioned using the Service Provider Group and Application Group SLA MBeans. See the discussion on Managing SLAs in *Services Gatekeeper Portal Developer's Guide*.

Properties for Credit Control Service Interceptors

Table 24–2 describes the properties of Credit Control Service Interceptors.

Table 24–2 Credit Control Service Interceptor Properties

Property	Description
Managed object in Administration Console	<*domain_name*> then OCSG then <server name> then ContainerServices then CreditControlInterceptor

MBean	Domain=com.bea.wlcp.wlng
	Name=wlng_nt
	InstanceName=CreditControlInterceptor
	Type=com.bea.wlcp.wlng.cc_interceptor.management.DiameterMBean

Deployment artifacts	cc_interceptor.ear

Deployment of CreditControlInterceptor

By default Credit Control interceptors are not deployed.

Deploy the EAR file that contains the interceptors using WebLogic Server tools. See "Java EE Deployment Implementation" in *Oracle WebLogic Server Deploying Applications to Oracle WebLogic Server* for a description of the different deployment options.

The EAR file is located in *Gatekeeper_Home*/`applications`.

Following is an example on how to the deploy the Credit Control Interceptors:

```
java weblogic.Deployer -adminurl http://<admin host>:<admin port> -user <admin
user> -password <admin password -name ccInterpeptor.ear -deploy
```

The interceptors are not connected to the Diameter server after deployment.

Configuring CreditControlInterceptor

To configure the behavior of the **CreditControlInterceptor**, in the managed object **CreditControlInterceptor**:

1. Specify entries for these fields:
   - **PeerRetryDelay**
   - **DestinationHost**
   - **DestinationPort**
   - **DestinationRealm**
   - **OriginHost**
   - **OriginPort**
   - **MoReservationInterceptorIndex**
   - **MtReservationInterceptorIndex**
   - **CommitInterceptorIndex**
   - **PeerRetryDelay**
   - **RequestTimeout**
   - **WatchdogTimeout**
2. Use the **connect** method to connect to the Diameter server.

Managing CreditControlInterceptor

The Credit Control Interceptors can be explicitly connected to the Diameter server. It does not connect to the server by default. The interceptors have a connection status that will be preserved after redeployment and server restart.

Use:

- The **connect** method
- The **disconnect** method

Use the **Connected** (r) field to check the current connection status.

Use the: **connect** method after any changes to the configuration attributes. Changes do not take affect until this operation is invoked.

Provisioning Credit Control SLAs

Credit Control SLAs can be enforced at the application group and service provider group level. These SLAs are provisioned as custom SLAs. See the discussion about managing SLAs in the *Services Gatekeeper Accounts and SLAs Guide*.

Before any Credit Control SLAs can be provisioned, the XSD for the SLA must be loaded, and the SLA type is given when the XSD is loaded.

The XSD is found in an **.ear** file that contains the Credit Control interceptors. See "Deployment of CreditControlInterceptor" for more information. The XSD is located in

```
/xsd/ccMapping.xsd
```

The SLA type to use when loading the Credit Control SLAs XSD is `credit_control`.

The SLA type to use when provisioning Credit Control SLAs is `credit_control`.

25

Implementing Diameter Rf Offline Charging

This chapter describes how to manage and configure the Oracle Communications Services Gatekeeper call detail record (CDR)-to-Diameter service for offline charging.

Oracle Communications Services Gatekeeper forwards CDR charging events to a Diameter Rf server for offline charging. Services Gatekeeper offline charging uses the Charging management; Diameter charging; 3GPP TS 32.299 V8.4.0 (2008-09) (Release 8), Diameter Credit-Control Application; RFC 4006 here:

`http://www.ietf.org/rfc/rfc4006.txt`

You can also use the Parlay X 3.0 Payment/Diameter communication service to establish direct billing. For details see "Parlay X 3.0 Payment/Diameter" in *Services Gatekeeper Communication Service Reference Guide*.

About CDRs and Diameter

The **CdrToDiameter** service forwards charging events to a Diameter server using the Diameter Rf interface. This allows for easy integration with charging systems that support Diameter Rf offline charging. When enabling this service, Oracle Communications Elastic Charging Engine or the Oracle Communications Billing and Revenue Management products, with Oracle Communications Network Mediation, can be used out-of-the-box with Services Gatekeeper.

Any traffic request processed through Services Gatekeeper that generates a CDR also generates a corresponding Diameter Rf request. The Diameter server can the be used for any billing, charging, and reporting for these requests. See "CDR to AVP mapping" for information about how CDRs are mapped to Diameter attribute-value pairs (AVPs).

CdrToDiameter Service Deployment Characteristics

The **CdrToDiameter** service is not deployed by default. It is deployed as a regular JEE module using WebLogic Server deployment tools. For information on how to deploy the service, see "Java EE 6 Deployment Implementation" in *Oracle Fusion Middleware Understanding Oracle WebLogic Server*.

The service is packaged in **cdr_to_diameter-single.ear** and **cdr_to_diameter.ear** in *Gatekeeper_home***/applications**.

Use **cdr_to_diameter.ear** in clustered installations, and **cdr_to_diameter-single.ear** in single server domains.

The service is a cluster singleton, so it will execute only on one server at any given time. In case of server failure, this service is transferred to another server. The management part is distributed to all servers in the cluster, so it can be managed from any server in the cluster.

Configuration of CdrToDiameter

To configure the behavior of the **CdrToDiameter** service, in the managed object **CdrToDiameter**:

1. Specify these fields:
   - **OriginHost**
   - **OriginPort**
   - **DestinationHost**
   - **DestinationPort**
   - **DestinationRealm**
   - **PeerRetryDelay**
   - **RequestTimeout**
   - **WatchdogTimeout**
2. Use the **connect** method to connect to the Diameter server.

Management of CdrToDiameter

The **CdrToDiameter** service can be explicitly connected to the Diameter server. It does not connect to the server by default. The service has a connection status that will be preserved after service redeployment and server restart.

Use these methods in **CDRDiameterMBean**:

- **connect**
- **disconnect**

Use **Enabled (r)** to check the current connection status.

Use **connect** after any changes to the configuration attributes. Changes does not take affect until this operation is invoked.

CDR to AVP mapping

The CDRs that are generated are mapped to Diameter attribute-value pairs, AVPs, as shown in Table 25–1. Both standard and custom AVPs are used. When custom are used it is indicated in the table.

Table 25–1 CDR to Diameter AVP Mappings

AVP	Source	Description
Session-Id	Auto generated.	Operation session identifier. Specification: RFC 3588 Example: nt1.ocsg.oracle.com;1214342798;0

Table 25–1 (Cont.) CDR to Diameter AVP Mappings

AVP	Source	Description
Origin-Host	Configurable, using the **OriginHost** field to **CDRDiameterMBean**	Realm of the Services Gatekeeper domain. Addressed with the domain address of the corresponding public URI. Specification: RFC 3588 Example: nt1.ocsg.oracle.com
Origin-Realm	Auto generated.	Ream of the Services Gatekeeper domain. Addressed with the domain address of the corresponding public URI. Specification: RFC 3588 Example: oracle.com
Destination-Realm	Configurable using the **DestinationRealm** field in **CDRDiameterMBean**	Realm of the operator domain. Addressed with the domain address of the corresponding public URI. Specification: RFC 3588 Example: oracle.com
Accounting-Record-Type	Value of CDR name-value pair: CallInfo	Defines the transfer type. Value is EVENT_RECORD for event based charging. Value is START_RECORD, INTERIM_RECORD, or STOP_RECORD for session based charging. If value of CallInfo is START, transfer type is EVENT_RECORD or START_RECORD. If value of CallInfo is STOP, transfer type is STOP_RECORD. EVENT_RECORD has numeric value 1. START_RECORD has numeric value 2. INTERIM_RECORD has numeric value 3. STOP_RECORD has numeric value 4. Specification: RFC 3588 Example: 1
Accounting-Record-Number	Auto generated.	Sequence number of the Diameter message sent from the CdrToDiameter service. Specification: RFC 3588 Example: 42
Acct-Application-Id	Static. Value is always 3.	Value is always 3, which corresponds to the application ID of the Diameter Accounting Application (off-line charging). Specification: RFC 3588

Table 25–1 (Cont.) CDR to Diameter AVP Mappings

AVP	Source	Description
User-Name	Value of CDR name-value pair: ServiceProviderId	The service provider ID associated with the request that triggered the CDR. Specification: RFC 3588 Example: service_provider_1
Event-Timestamp	CDR attribute in name-value pair: Timestamp	Time the event happened. Given in seconds since the year 1900. Specification: RFC 3588
Calling-Party-Address	Value of CDR name-value pair: destinationParty	Custom AVP. Depending on which communication service that triggered the CDR, different application-provided parameters are used. Specification: 3GPP 32.299 Example: tel:7878
Called-Party-Address	Value of CDR name-value pair: originatingParty	Custom AVP. Depending on which communication service that triggered the CDR, different application-provided parameters are used. Specification: 3GPP 32.299 Example: tel:7878
Service-Indication	Value of CDR name-value pair: ServiceName	Custom AVP. The name of the service the request triggered. Specification: 3GPP 29.329 Example: MultimediaMessaging
Message-Size	Calculated from message payload. Relevant for Parlay X 2.1 Short Messaging and Multimedia Messaging.	Custom AVP. For MM, it holds the total size in bytes of the MM calculated according to TS 23.140 For SM, it holds the total size in octets of the SM including any user data header. Specification: 3GPP 32.299 Example: 345
OCSG-Charge-Description	Parameter ChargingDescription in any Parlay X operation that includes this parameter.	Custom AVP. Application-provided description text to be used for information and billing text. Specification: none Example: Delivery of weather report.

Table 25–1 (Cont.) CDR to Diameter AVP Mappings

AVP	Source	Description
Currency-Code	Parameter ChargingCurrency in any Parlay X operation that includes this parameter.	Custom AVP. Currency code mapped according to ISO 4217. Specification: RFC 4006 Example: If the currency is U.S dollars, the value is 840
Unit-Value.Exponent	Parameter ChargingAmount in any Parlay X operation that includes this parameter.	Custom AVP. The exponent value to be applied for the Value-Digit AVP within the Unit-Value AVP. Specification: RFC 4006 Example: -2
Unit-Value.Value-Digits	Parameter ChargingAmount in any Parlay X operation that includes this parameter.	Custom AVP. The value to be applied for the Value-Digit AVP within the Unit-Value AVP. Contains the significant digits of the number scaled to an integer. Specification: RFC 4006 Example: 4062
Service-Context-Id	Parameter ChargingCode in any Parlay X operation that includes this parameter.	Custom AVP. Code that references a contract under which the charge is applied. Specification: RFC 4006 Example: premium

26

Managing and Configuring Native UCP Connections

This chapter describes how to create and store connection and credential mappings in the Oracle Communications Services Gatekeeper Connection Information Manager. The Connection Information Manager is used by the native UCP plug-ins.

Understanding the Connection Information Manager

The Connection Information Manager creates and stores connection and credential mappings that some plug-in instances need to connect to network elements and applications.

The **ConnectInfo** object stores the following connection information:

- The remote address to which the plug-in connects
- A map of application instance IDs (for example for Services Gatekeeper users) to network node credentials; there could be more than one set of credentials for a single network node

See the **getConnectInfo** method in **ConnectInfoManagerMBean** for details about the **ConnectInfo** object.

Configuring and Managing the Connection Information Manager

All configuration and management is performed in the **ConnectionInfoManager** managed object.

This object is accessible from the Administrative Console and from the Platform Test Environment (PTE).

The Administration Console displays one instance of the MBean for each plug-in instance that uses it. The management attributes for the Connection Information Manager are reached from the service:

- **Plugin_sms_ucp** then **ConnectInfoManager**

The Connection Information Manager Operations, Administration, and Maintenance functionality is shared among the plug-ins. Rendering, however, is per plug-in (one instance is displayed per plug-in). The appearance may vary depending on how you access the MBean.

If you are using different plug-ins that use the Connection Information Manager, you must configure the Connection Information Manager for each plug-in.

Use the following **ConnectInfoManagerMBean** methods to configure and manage connection information:

- **createOrUpdateLocalHostAddress**
- **createOrUpdateRemoteHostAddress**
- **getConnectInfo**
- **removeConnectInfo**
- **removeLocalHostAddress**

Use the following **ConnectInfoManagerMBean** methods to configure and manage the credential map:

- **addXParamToCredentialEntry**
- **createOrUpdateCredentialMap**
- **createOrUpdateUserPasswordCredentialEntry**
- **listAllCredentialEntries**
- **removeCredentialEntry**
- **removeCredentialMap**

Use the following **ConnectInfoManagerMBean** methods to configure and manage the listen information:

- **createOrUpdateListenAddress**
- **getAllListenAddress**
- **removeListenAddress**
- **getListenAddressForCurrentServer**

ConnectionInfoManagerMBean Reference

To set field values and use methods from the Administration Console, select **Container Services**, then **ConnectionInfoManagerMBean**. Or use a Java application. For information on the methods and fields, see the “All Classes” section of *Services Gatekeeper OAM Java API Reference*.

27

Managing and Configuring Parlay X 2.1 Shortcode Mappings

This chapter describes how to configure shortcode mappings for mobile-originated traffic to Oracle Communications Services Gatekeeper.

Understanding the Shortcode Mapper

The shortcode mapper is shared between the following plug-ins:

- Parlay X 2.1 Short Messaging/SMPP
- Parlay X 2.1 Multimedia Messaging/MM7

The shortcode mapper maps network-triggered messages (mobile originated) with a given destination address, the *original destination address*, to another destination address, the *translated destination address*.

The original destination address can be expressed as a pattern (a regular expression), which means that a range, or set, of original destination addresses can be translated to one single translated destination address.

This is useful when applications are triggered by a range of phone numbers; for example, 2345600 through 2345699. Using the functionality available in the Parlay X 2.1 Short Messaging interface, the application would have to call `startSmsNotification` 100 times. Shortcode mapping allows the application to express the original destination address as a pattern, such as 23456?? generating only a single call to `startSmsNotification` using this address.

Configuring and Managing the Shortcode Mapper

All configuration and management is performed in the **ShortCodeMapper** managed object.

The **ShortCodeMapper** operations, administration and management (OAM) WebLogic interface functionality is shared among the plug-ins: they reuse the same MBean. Rendering, however, is per plug-in (one instance is displayed per plug-in). The **ShortCodeMapper** appears slightly differently depending on how to the MBean is accessed:

- In the Administration Console, expand the plug-in that uses short code mapping:
  - **Plugin_px21_multimedia_messaging_mm7** then **ShortCodeMapper**
  - **Plugin_px21_short_messaging_smpp** then **ShortCodeMapper**

Then click **ShortCodeMapper** to display the operations and attributes for the shortcode mapper.

- In an MBean browser, such as JConsole, one instance of the MBean is displayed using the same ObjectName (at the same hierarchical level) as the plug-in it is used by.

> **Note:** You must configure **ShortCodeMapper** attributes for *all* of the above mentioned plug-ins that are going to use this functionality.
> Only the attributes for the related plug-in are displayed in the console.

Management operations

You manage short codes using these **ShortCodeMapperMBean** methods:

- **addShortCodeMapping**
- **listShortCodeMappings**
- **removeShortCodeMapping**

For a description of the attributes and operations of the **ShortCodeMapperMBean** MBean, see *Oracle Communication Services Gatekeeper OAM Java API Reference*.

28

Managing OSA/Parlay Gateway Connections Using Parlay_Access

This chapter describes Open Services Architecture (OSA)/Parlay Gateways and explains how to connect them to Oracle Communications Services Gatekeeper.

Understanding OSA/Parlay Gateway and account mappings

This section describes the general model Services Gatekeeper uses to deal with OSA/Parlay gateways.

Connection model

Services Gatekeeper communication services use an internal service, Parlay Access, to manage all connections with OSA/Parlay Gateways. A plug-in that connects to an OSA/Parlay Service Capability Server (SCS) asks the OSA Access service for a connection, and the service handles all of the details of authentication, service discovery, and load management toward the OSA/Parlay framework before returning the handle for the SCS to the plug-in.

The following concepts relate to a plug-in connected to an OSA/Parlay Gateway:

- An OSA/Parlay Gateway is identified by a **gatewayId**, which represents the actual OSA/Parlay Gateway. Each OSA Gateway used is registered in Services Gatekeeper. Any certificate to be used when authenticating with the OSA/Parlay framework is associated with the **gatewayId**.
- Each OSA/Parlay Gateway has one or more **OSA/Parlay Gateway Connections**, identified by a **connectionID**. Multiple connections are used if the actual OSA/Parlay Gateway contains more than one Framework. The link between the OSA Gateway and the OSA Gateway connection is the **gatewayID/gwID**.
- An **OSA/Parlay client** represents the account in the OSA/Parlay Gateway. An OSA client has the following attributes:
  - OSA client application ID, made up of the Enterprise Operator ID and the Application ID as provisioned in the OSA/Parlay Gateway,
  - Depending on the authentication method used, a private key (with associated password and keystore password) and public certificate to be used when authenticating.
- An **OSA/Parlay client mapping** maps an OSA/Parlay client with OSA/Parlay SCSes. There must be at least one OSA/Parlay client mapping per OSA SCS being used. If the communication service uses *n* OSA/Parlay SCSes, *n* client mappings

must be defined. Three different models are possible for the OSA/Parlay client mapping:

- The client mapping can use wild cards for both the service provider and the application level, so all applications from all service providers are mapped to a single Client. In this case, transactions in the OSA/Gateway are traceable only to Services Gatekeeper because Services Gatekeeper, from the OSA/Parlay Gateway’s viewpoint, acts as one single application.
- The client mapping can use a wildcard for the application level and specify the service provider, so multiple Services Gatekeeper applications that originate from a common service provider are mapped to a single OSA client. In this case, the transactions in the OSA/Gateway are traceable only to the service provider because Services Gatekeeper, from the OSA/Parlay Gateway’s viewpoint, acts as one application per service provider.
- The mapping may be set up per application level, so there is a one-to-one mapping between an Services Gatekeeper service provider and application account combination and the equivalent client. This means that every transaction originating from a specific application results in a transaction in the OSA/Parlay Gateway that is traceable to that specific application because Services Gatekeeper, from the OSA/Parlay Gateway’s viewpoint, acts as one application per service provider and application combination.

Note: Combinations of the above are not allowed. The Services Gatekeeper administrator must choose one of these connection modes and use the same mode for all Services Gatekeeper applications. In the first case, the connection is a systemwide configuration. In the other two cases, the connection is set up as a part of the provisioning chain for Services Gatekeeper service providers and their applications.

Defining the OSA/Parlay client mapping is a part of the provisioning chain in when setting up service provider and application accounts if the client mapping is of type b. or type c.

Each OSA/Parlay Client mapping has a state:

- **Active**, when the connection between Services Gatekeeper and a specific SCS in the OSA/Parlay Gateway is active and functional.
- **Inactive**, when there is no active connection. This may be because the client mapping is not configured to be initialized at startup and no requests have yet been passed to it. It may also indicate that there is a problem with the connection.

Information and Certificate Exchange with OSA/Parlay Gateway Administrator

The OSA/Parlay Gateway administrator must provide the following information with regard to the OSA/Parlay Gateway account and OSA/Parlay Framework:

- The **entOpId** (Enterprise Operator ID): Depending on how the OSA/Parlay operator administers applications (OSA/Parlay clients), the entOpId can be valid for:
  - All applications registered in Services Gatekeeper
  - All applications connected to a service provider account
  - A single application account

- The **appId** (Application ID) to be used for the application account; used with the **entOpId** in **clientAppId** parameters to various operations
- The OSA/Parlay **service types** for the OSA/Parlay SCSes to which the application is to be mapped
- The encryption method used
- The signing algorithm used
- Connection information for the OSA/Parlay Framework, either:
  - name service reference file to the OSA/Parlay Gateway Framework's Parlay IpInitial object.
  - name of the initial object in the name service and the file containing the Interoperable Object Reference (IOR) to the **IpInitial** object.
- If the authentication method toward the OSA/Parlay Framework requires a certificate, the Services Gatekeeper administrator must generate one and distribute it to the OSA/Parlay Gateway administrator. The associated key must be stored in the Services Gatekeeper keystore. This is done when the OSA client is created: see "Creating an OSA client" for details.

  For non-production environments, the WebLogic Server **CertGen** utility can be used to create certificates and keys.

Connecting to an OSA Gateway

To connect an application account to an OSA/Parlay Gateway:

1. Create a logical representation of the OSA/Parlay Gateways to connect to. See "Adding an OSA/Parlay Gateway" for details.
2. For each Framework in the OSA/Parlay Gateway, create a logical representation of the Framework. See "Adding an OSA Gateway Connection".
3. Define how Services Gatekeeper connects to the OSA/Parlay Gateway.
   - If Services Gatekeeper connects to the OSA/Parlay Gateway as one single user, register this user. See "Creating an OSA client".
   - If Services Gatekeeper connects to the OSA/Parlay Gateway as several users, the registration of users is a part of the provisioning flow for service providers and applications.
4. The registration of which SCSes to use in the OSA/Parlay Gateway is done either as a part of the configuration flow for the communication services or as a part of the provisioning flow for service providers and application. The procedure is described in "Mapping the OSA client to an OSA Gateway and an OSA/Parlay SCS", and the data to be used is described in the configuration section for each communication service.

Adding an OSA/Parlay Gateway

An OSA/Parlay Gateway connection is the entity representing an OSA/Parlay Gateway. One or more OSA Gateway Connections can be associated with the OSA Gateway.

1. If authenticating using certificates, get the certificate for the OSA/Parlay Gateway from the administrator of the OSA/Parlay Gateway and store it on the local file system of the Services Gatekeeper administration server.

2. Starting in the configuration and operations page for Plugin_Parlay_Access_ *communication service*, select **addGw** from the **Select An Operation** list.

   The parameters for the operation are displayed.

3. Enter the gateway information using the **ParlayAccessMBean addGw** method.

4. Click **Invoke**.

   The OSA Gateway is created. An ID for the OSA Gateway is returned.

Adding an OSA Gateway Connection

An OSA Gateway connection is the entity representing an individual Framework in an OSA/Parlay Gateway.

1. Get information about how to obtain a reference to the OSA/Parlay Framework from the administrator of the OSA/Parlay Gateway. These options are possible:

   – The name service reference file. Store the file on the local file system of the Services Gatekeeper administration server.

   – The name of the initial object in the name service and the file containing the IOR to the Parlay initial object. Store the file on the local file system of the Services Gatekeeper administration server.

   – The IOR is provided as a String.

2. Starting in the configuration and operations page for Plugin_Parlay_Access_ *communication service*:

   If the IOR is provided as a file use the addConnection method

   If the IOR is provided as a String use the AddConnectionIOR method

3. Click **Invoke**.

   The OSA Gateway Connection is created. An ID for the OSA Gateway Connection is returned.

Creating an OSA client

The OSA client is the entity being used when creating the OSA client mapping.

1. If you are authenticating using certificates, create, or get from a Certificate Authority, the private key and certificate for the client and store them on the local file system of the Services Gatekeeper administration server.

2. Starting in the configuration and operations page for Plugin_Parlay_Access_ *communication service*, select **addClient** from the **Select An Operation** list.

   The parameters for the operation are displayed.

3. Enter client information using the **ParlayAccessMBean addClient** method.

4. Click **Invoke**.

   The OSA client is created.

Mapping the OSA client to an OSA Gateway and an OSA/Parlay SCS

The mapping may be applied on service provider account, application account, or Services Gatekeeper level.

Note: One mapping must be created for each OSA/Parlay SCS (network service) the Services Gatekeeper application is using in the OSA/Parlay Gateway.

1. Starting in the configuration and operations page for Plugin_Parlay_Access_*communication service*, select **addMapping** from the **Select An Operation** list.

   The parameters for the operation are displayed.

2. Enter mapping information using the **ParlayAccessMBean addMapping** method.
3. Click **Invoke**.

   The OSA client mapping is created.

For a description of the attributes and operations of the **ParlayAccessMBean** MBean, see *Oracle Communication Services gatekeeper OAM Java API Reference*.

29

Managing Legacy Application Service Providers

This chapter explains the Oracle Communications Services Gatekeeper framework for managing service providers and applications that was used before the partner relationship manager (PRM) portals were created.

This functionality is an alternative to using the PRM portals to create partners, partner groups, and network service suppliers (NSSs). This chapter has the same concepts but a different nomenclature. It explains how to manage the *service providers* and *service provider groups* that perform many of the same functions as partners and partner groups.

WARNING: Use either the PRM portals to create these entities, or the instructions in this chapter, but do not attempt to mix them.

About the Management Framework

Services Gatekeeper provides a partner administration model that you can use to manage your partner relationships. Application service providers are registered with Services Gatekeeper through service provider accounts and application accounts. Each account type is associated with a group. Each group is associated with an SLA that defines its access to both Services Gatekeeper and underlying network nodes.

You can register service providers in the following ways:

- Using the Services Gatekeeper Administration Console graphical user interface.
- Using an external management system, integrated with Services Gatekeeper using the partner relationship management graphical user interface.
- Using an external management system integrated with Services Gatekeeper using JMX.

The Administration Model

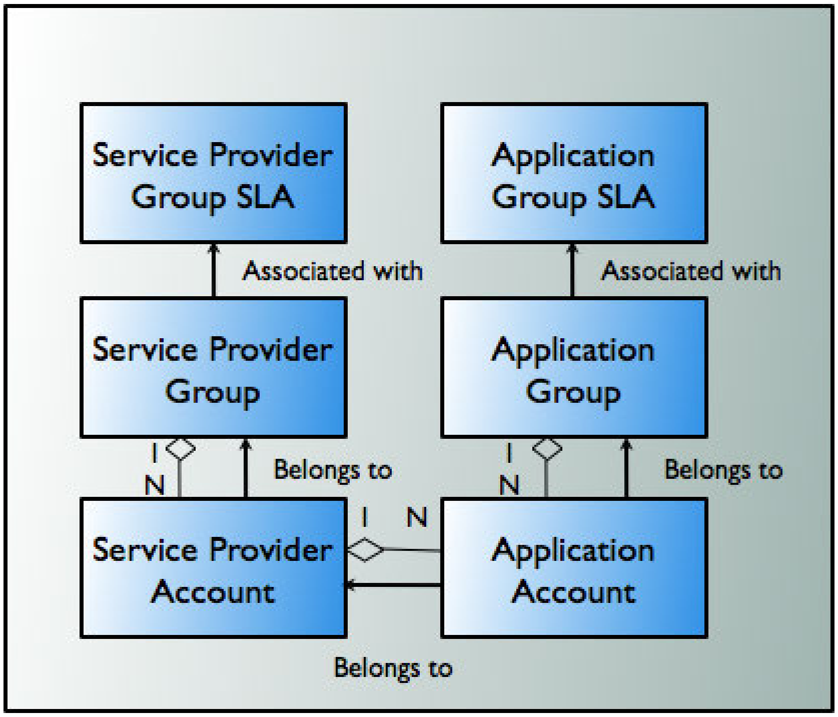
The Services Gatekeeper administration model enables you to control application-service-provider access at various levels. An application service provider registers with Services Gatekeeper and is given a service provider account. To support tiering, you assign service provider accounts to account groups. You then associate these account groups with their own service provider group SLAs.

Individual application accounts are registered within a service provider account. These application accounts also belong to account groups, each of which is associated with its own account group SLA.

Service provider group and account group SLAs regulate, for example, the type of service capability made available and the maximum bandwidth use allowed. SLAs may also specify access to charging capabilities and revenue sharing schema. Services Gatekeeper provides standard versions of service provider group and application group SLAs, and custom SLAs of both types.

Figure 29–1 illustrates the relationship between accounts, groups, and their SLAs.

Figure 29–1 Service Provider and Application Administration Model

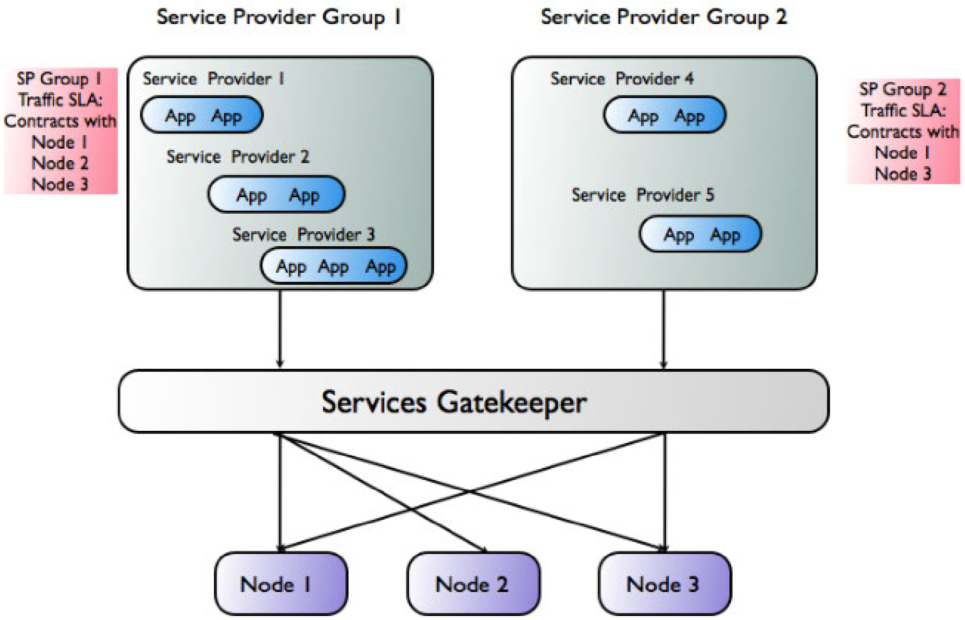


Using the Platform Development Studio, you can extend this model to include users as well. For more information, see *Services Gatekeeper Extension Developer's Guide*.

In addition to the service provider group and account group SLAs, Services Gatekeeper provides two types of traffic SLAs: service provider node SLAs and global node SLAs. You can also create global node custom SLAs. These are contracts designed to protect the underlying telecom network. Service provider node SLAs regulate the relationship between a service provider group and the network nodes to which it has access.

Figure 29–2 illustrates the relationship between service provider groups and network nodes, based on the service provider node SLAs.

Figure 29–2 Service Provider Traffic SLAs



In Figure 29–2, service providers in service provider group 1 are allowed to access all network nodes because their service provider node SLA (valid for all service providers within the group) contains node contracts for all nodes.

Service providers in service provider group 2 are allowed to access only network nodes 1 and 3 because their service provider node SLA contains contracts only for nodes 1 and 3.

The second type of traffic SLA, the Global Node SLA, regulates the overall relationship between Services Gatekeeper and the underlying nodes.

Partner Relationship Management Interfaces

You use the Services Gatekeeper partner relationship management interfaces to automate tasks related to service provider and application administration. These interfaces support request and approval workflow for helping service providers set up their service provider and application accounts. The interfaces also enable service providers to communicate account change requests, and to retrieve usage statistics for their accounts.

For a detailed description of the partner relationship management interfaces, see *Services Gatekeeper Portal Developer's Guide*.

Note: For an application to use Audio Call-based services, announcements must be recorded and installed in the network.