Oracle® Communications Services Gatekeeper

System Administrator’s Guide

Release 5.1

E37531-01

June 2013

Oracle Communications Services Gatekeeper System Administrator's Guide, Release 5.1

E37531-01

Contents

Preface xiii

Audience xiii
Documentation Accessibility xiii
Related Documents xiii

1 Introduction

Container Services 1-1
Overall Configuration Workflow 1-1

2 Starting and Stopping Servers

Using Scripts 2-1
Using Node Manager 2-2
HP-UX 2-3
Solaris 64-bit 2-3

3 Operation and Maintenance: General

Administration Console Overview 3-1
Starting and Using the Administration Console 3-1
MBeans pane 3-2
Configuration and Provisioning 3-3
WebLogic Oracle Communications Services Gatekeeper Alarms Pane 3-5
Oracle Communications Services Gatekeeper CDRs Pane 3-7
Converged Application Server Administration Console for SIP-based Services 3-8
Java Management Extensions 3-8
WebLogic Scripting Tool (WSLT) 3-11
Working in Interactive Mode 3-12
Starting WLST and connecting to Services Gatekeeper 3-12
Exiting WLST 3-12
Changing an attribute 3-12
Invoking an operation 3-13
Scripting WLST 3-13
Database Maintenance 3-14
About Cleaning Database Tables 3-14
About Cleaning EDR Analytics Tables 3-15
About Cleaning the SLEE_CHARGING Table 3-15

Creating a Script to Clean SLEE_CHARGING 3-15

4 Managing Management Users and Management User Groups

Task Overview 4-1
Users and User Groups 4-1
User Types 4-2
User Level 4-3
Reference: Attributes and Operations for ManagementUsers 4-4
Operation: addUser 4-4
Operation: changeUserPassword 4-4
Operation: deleteUser 4-5
Operation: getUserLevel 4-5
Operation: listUsers 4-5
Reference: Attributes and Operations for ManagementUserGroup 4-6
Operation: addUsertoGroup 4-6
Operation: createUserGroup 4-6
Operation: listGroups 4-7
Operation: listUsers 4-7

5 Managing and Configuring SOA Facades

Introduction 5-1
Upgrading SOA Facade Projects From Previous Release 5-2
Loading SOA Facade Projects 5-2
Available SOA Facades 5-3

6 Configuring Coherence

About Configuring Coherence 6-1
Configuring Coherence 6-1

7 Managing and Configuring Budgets

Introduction 7-1
Synchronization of budgets 7-2
Slave intervals 7-2
Master Internal 7-3
Failure Conditions 7-3
Budget Overrides 7-3
Budget Calculations and Relationship to SLA Settings 7-3
Configuration and Management 7-5
Reference: Attributes and Operations for BudgetService 7-5
Attribute: PersistentBudgetFlushInterval 7-5
Attribute: PersistentBudgetTimeThreshold 7-5
Attribute: AccuracyFactor 7-5
Attribute: ConfigUpdateInterval 7-6
Adding a Datasource 7-6

8 Managing and Configuring EDRs, CDRs and Alarms

About EDRs, CDRs, and Alarms 8-1
EDR categories and XML markup 8-1
EDR format 8-2
EDRs 8-4
Alarms 8-4
CDRs 8-4
External EDR listeners 8-4
EDRService 8-4
Configuration of the EDRService 8-4
Management of the EDRService 8-4
Defining batch attributes 8-4
Reference: Attributes and Operations for EDRService 8-4
Attribute: BatchTimeout 8-5
Attribute: StatisticsEnabled 8-5
Attribute: BatchSize 8-5
Attribute: PublishToJMS 8-5
Attribute: StoreAlarms 8-5
Attribute: StoreCDRs 8-5
Operation: displayStatistics 8-6
Operation: resetStatistics 8-6
Operation setEdrType 8-6
Managing EDR, CDR, and alarms configuration files using the EDR Configuration Pane 8-6

9 Managing and Configuring Credit Control Interceptors and SLAs

Introduction 9-1
Application-Initiated Requests 9-1
Network-Triggered Requests 9-1
Credit Control Interception Points 9-1
Writing Credit Control SLAs 9-2
Defining Static and Dynamic Parameter Mappings 9-6
Correlating Reservation and Commit Triggers in Asynchronous Credit Control Checks 9-6
Example Credit Control SLA 9-7
Configuration, Management and Provisioning 9-8
Properties for Credit Control Service Interceptors 9-8
Deployment of CreditControlInterceptor 9-8
Configuration of CreditControlInterceptor 9-9
Management of CreditControlInterceptor 9-9
Provision Credit Control SLAs 9-9
Reference: Attributes and Operations for CreditControlInterceptor 9-10
Attribute: Connected (r) 9-10
Attribute: CommitInterceptorIndex 9-10
Attribute: DestinationHost 9-10
Attribute: DestinationPort 9-11
Attribute: DestinationRealm 9-11
Attribute: OriginHost 9-11

Attribute: OriginPort 9-11
Attribute: MoReservationInterceptorIndex 9-11
Attribute: MtReservationInterceptorIndex 9-11
Attribute: PeerRetryDelay 9-12
Attribute: RequestTimeout 9-12
Attribute: WatchdogTimeout 9-12
Operation: connect 9-12
Operation: disconnect 9-12

10 Setting Up Geographic Redundancy

Introduction 10-1
Configuration Workflow 10-1
Configure Each Site for Geo-Redundancy 10-1
Define the GeoMaster Site 10-1
Reference: Attributes and Operations for GeoRedundantService 10-2
Attribute: GeoSiteId 10-2
Attribute: RemoteSiteReachabilityAlarmThreshold 10-2
Operation: getSiteAddress 10-2
Operation: listRemoteSites 10-2
Operation: removeSite 10-3
Operation: setSiteAddress 10-3
Reference: Attributes and Operations for GeoStorageService 10-3
Attribute: GeoMasterSiteId 10-3
Operation: syncFromGeoMaster 10-4

11 Managing and Configuring Statistics and Transaction Licenses

About Statistics Generation and Reports 11-1
Overview of Statistics Reports 11-1
System Report to Console 11-2
System Report to File 11-2
Weekly System Report 11-2
Transaction Usage Log Report 11-2
Counter Snapshots 11-3
Managing StatisticService 11-4
Configure the StatisticService 11-4
Configure Statistics Types and Transaction Types 11-4
View In-Flight Statistics counters 11-4
Generate Statistics Reports 11-5
Add Usage Thresholds 11-5
Reference: attributes and operations for StatisticsService 11-5
Attribute: CounterSnapshot (r) 11-6
Attribute: StoreInterval 11-6
Attribute: ModuleBHTUPSThreshold 11-6
Attribute: PlatformBHTUPSThreshold 11-6
Operation: addStatisticType 11-6
Operation: createLicenseLimitLog 11-7
Operation: createWeeklyReport 11-7

Operation: getStatistics 11-8
Operation: getSystemStatistics 11-8
Operation: listStatisticTypeDescriptors 11-9
Operation: listStatisticTypes 11-9
Operation: removeStatisticType 11-9
Operation: saveAccountStatisticsToFile 11-9
Operation: saveStatisticsToFile 11-10
Transaction Types 11-11

12 CDRs and Diameter

About CDRs and Diameter 12-1
CdrToDiameter Service Deployment Characteristics 12-1
Configuration of CdrToDiameter 12-2
Management of CdrToDiameter 12-2
Reference: Attributes and Operations for CdrToDiameter 12-2
Attribute: Enabled (r) 12-3
Attribute: OriginHost 12-3
Attribute: OriginPort 12-3
Attribute: DestinationHost 12-3
Attribute: DestinationPort 12-3
Attribute: DestinationRealm 12-3
Attribute: PeerRetryDelay 12-4
Attribute: RequestTimeout 12-4
Attribute: WatchdogTimeout 12-4
Operation: connect 12-4
Operation: disconnect 12-4
CDR to AVP mapping 12-5

13 Managing and Configuring the SNMP Service

Introduction 13-1
Configuration and management 13-2
Configure SNMPService 13-2
Trap Receivers 13-2
Reference: Attributes and Operations for SNMPService 13-2
Attribute: Community 13-3
Attribute: EnterpriseObjectIdentifier 13-3
Attribute: RepeatedTraps 13-3
Attribute: SNMPVersion 13-3
Attribute: SeverityFilter 13-4
Operation: addTrapReceiver 13-4
Operation: deleteTrapReceiver 13-4
Operation: listTrapReceivers 13-4

14 Managing and Configuring the Trace Service

Introduction to the Trace Service 14-1
Basic tracing 14-1

Context trace 14-1
Reference: Attributes and Operations for Trace Service 14-2
Attribute: TracingEnabled 14-2
Operation: addContextCategory 14-2
Operation: addContextFilter 14-3
Operation: attachAppender 14-3
Operation: createContextTraceFile 14-4
Operation: createRootContextTraceFile 14-4
Operation: flushBuffers 14-5
Operation: removeContextTraceFile 14-5
Operation: resetContextFilters 14-5
Operation: rollOver 14-5
Log4J Hierarchies, Loggers, and Appenders 14-6
Configuring Trace for Access Tier servers 14-6
Using the Log4J Configuration File 14-7
Example Log4J Configuration file 14-7

15 Managing and Configuring the Storage Service

Introduction to the Storage Service 15-1
Storage Data Expiration 15-2
Configuring the Expiration Period 15-2
Configuring the Checking Interval 15-3
Reference: Attributes and Operations for Storage Service 15-3
Attribute: Active (r) 15-3
Operation: getProviderName 15-3
Operation: getStoreSize 15-4
Operation: getTableName 15-4
Operation: getTypeId 15-4
Operation: listStoreNames 15-5

16 Securing Web Services and OAM MBeans

Introduction 16-1
Understanding Web Services Security 16-1
Understanding OAM MBean Security 16-2
Securing SOAP-Based Web Services with WS-Security 16-2
Setting up UsernameToken with Password Digest 16-3
Setting up UsernameToken with X.509 16-4
Removing Outbound Web Security 16-5
Removing WS-Policy from a Web Service 16-5
Create and use a custom WS-Policy 16-6
Available default WS-Policies 16-6
Securing RESTful Web Services with SSL 16-6
Configuring Application-Facing Servers for SSL 16-7
Enable and Configure SSL for Each Application Tier Server 16-7
Add Certificates to the Application Tier Servers and Applications 16-8
Securing Network-Facing Servers With Keystores 16-9
Securing the Oracle Access Manager MBeans 16-11

Administrative Groups 16-11
Administrative Service Groups 16-11

17 Deployment Model for Communication Services and Container Services

Packaging of Communication Services 17-1
Example 17-1
Deployment of SOAP and RESTful Facades on Multiple AT Clusters 17-3
Version Handling of Communication Services 17-5
Deploy and Undeploy Communication Services and plug-ins 17-5
Version Handling and Patching of Communication Services 17-5
Patch Tool 17-6
Examples 17-7
Overview of Container Services and Configuration Files 17-7
Container services 17-8
Patching of Container Services 17-9

18 Hitless Upgrade Using Production Redeployment

Production Redeployment Overview 18-1
Production Redeployment Sequence 18-2
Requirements for Production Redeployment 18-2
Typical Scenarios for Production Redeployment 18-2
Performing a Hitless Upgrade 18-4

19 Managing and Configuring the Plug-in Manager

Introduction 19-1
Execution and evaluation flow 19-2
Application-initiated requests 19-2
Network-triggered Requests 19-2
Plug-in Routing Logic 19-2
Defining Routing Logic 19-3
Plug-in Routing Configuration Examples 19-6
Plug-in Routing XSD 19-7
How address ranges are specified when setting up routes 19-8
Plug-in Routing Logic and Plug-in Routes 19-8
Configuration Workflow 19-8
Configuring the Plug-in Manager 19-9
Creating a Plug-in instance 19-9
Administration of Plug-in Routing Logic and Node IDs 19-9
Reference: Attributes and Operations for PluginManager 19-9
Attribute: ForceConnectInResuming 19-10
Attribute: PolicyBasedRouting 19-10
Attribute: supportBulkRequest 19-11
Operation: addRoute 19-11
Operation: createPluginInstance 19-11
Operation: destroyPluginInstance 19-12
Operation: getPluginInstanceInfo 19-12

Operation: getPluginNodeId 19-12
Operation: getPluginServiceInfo 19-13
Operation: getRouteConfig 19-13
Operation: getServiceInfo 19-13
Operation: listPluginInstances 19-14
Operation: listPluginServices 19-14
Operation: listRoutes 19-14
Operation: listServiceTypes 19-14
Operation: removeRoute 19-15
Operation: setRouteConfig 19-15
Operation: setPluginNodeId 19-15
Operation: listInterceptors 19-16
Operation: retrieveInterceptorConfiguration 19-16
Operation: updateInterceptorConfiguration 19-16

20 Managing and Configuring the Tier Routing Manager

Introduction 20-1
Configuration Workflow 20-2
Reference: Attributes and Operations for TierRoutingManager 20-2
Operation: addTierRoute 20-3
Operation: listTierRoutes 20-3
Operation: removeTierRoute 20-3

21 SOAP2SOAP Communication Services

About SOAP2SOAP Communication Services 21-1
Generating and Deploying SOAP2SOAP Communication Services 21-1
Generate a Communication Service 21-2
Deploy a Communication Service 21-3
Generated Artifacts for the Communication Service 21-4
Managing and Configuring SOAP2SOAP Communication Services 21-4
Properties for SOAP2SOAP Plug-ins 21-5
Configuration Workflow for SOAP2SOAP Plug-ins 21-6
Provisioning Workflow for SOAP2SOAP Communication Services 21-7
Reference: Attributes and Operations for SOAP2SOAP Plug-ins 21-7
Attribute: HttpBasicAuthentication 21-8
Attribute: HttpTimeoutPeriod 21-8
Attribute: HttpTimeoutThreshold 21-8
Attribute: HttpWaitTime 21-8
Attribute: NetworkEndpoint 21-8
Attribute: Password 21-9
Attribute: ReactivateTime 21-9
Attribute: UserName 21-9
Attribute: UserNameTokenAuthentication 21-9
Operation: addApplicationEndPoint 21-9
Operation: getApplicationEndPoint 21-10
Operation: listApplicationEndPoints 21-10
Operation: removeApplicationEndPoint 21-10

22 Configuring Heartbeats

Introduction 22-1
Configuration and Management 22-1
Reference: Attributes and Operations for HeartbeatConfiguration 22-2
Attribute: ContentMatch 22-2
Attribute: Enabled 22-2
Attribute: HeartbeatMethod 22-3
Attribute: Interval 22-3
Attribute: NetworkServiceUrl 22-3
Attribute: PostContent 22-3
Attribute: ResponseCodeToMatch 22-3
Attribute: Status (r) 22-4
Operation: addHTTPHeader 22-4
Operation: listHTTPHeaders 22-4
Operation: removeHTTPHeader 22-4

23 Managing and Configuring Connection Information

Introduction 23-1
Configuration and Management 23-1
Reference: Attributes and Operations for Connection Information Manager 23-2
Attribute: ValidationEnabled 23-3
Operation: addXParamToCredentialEntry 23-3
Operation: createOrUpdateCredentialMap 23-3
Operation: createOrUpdateListenAddress 23-4
Operation: createOrUpdateLocalHostAddress 23-4
Operation: createOrUpdateRemoteHostAddress 23-5
Operation: createOrUpdateUserPasswordCredentialEntry 23-5
Operation: getAllListenAddress 23-6
Operation: getConnectInfo 23-6
Operation: getListenAddressForCurrentServer 23-6
Operation: listAllCredentialEntries 23-7
Operation: removeConnectInfo 23-7
Operation: removeCredentialEntry 23-7
Operation: removeCredentialMap 23-7
Operation: removeListenAddress 23-8
Operation: removeLocalHostAddress 23-8
Operation: removeXParamFromCredentialEntry 23-8

24 Managing and Configuring Shortcode Mappings

Introduction 24-1
Configuration and Management 24-1
Management operations 24-2
Reference: Attributes and Operations for ShortCodeMapper 24-2
Operation: addShortCodeMapping 24-2
Operation: listShortCodeMappings 24-3
Operation: removeShortCodeMapping 24-3

25 Managing OSA/Parlay Gateway Connections using Parlay_Access

Understanding OSA/Parlay Gateway and account mappings 25-1
Connection model 25-1
Information and Certificate Exchange with OSA/Parlay Gateway Administrator 25-2
Overall workflow when connecting to an OSA Gateway 25-3
Adding an OSA/Parlay Gateway 25-3
Adding an OSA Gateway Connection 25-4
Creating an OSA client 25-4
Mapping the OSA client to an OSA Gateway and an OSA/Parlay SCS 25-4
Reference: Attributes and Operations for Parlay_Access 25-5
Attribute: EricssonAuthentication 25-6
Operation: activateMapping 25-6
Operation: addClient 25-6
Operation: addConnection 25-7
Operation: addConnectionIOR 25-7
Operation: addGw 25-8
Operation: addMapping 25-8
Operation: listActiveMappings 25-9
Operation: listActiveMappingsForGw 25-10
Operation: listGw 25-10
Operation: listMappings 25-10
Operation: removeClient 25-10
Operation: removeConnection 25-11
Operation: removeGw 25-11
Operation: removeMapping 25-11
Operation: setKeyStorePassword 25-11
Operation: viewActiveMappingState 25-12

26 Resolving Policy Deny Codes

Introduction 26-1
Policy Deny Data 26-1

27 Filtering Tunneled Parameters

About the XParameter Filter Application 27-1
XParameter Filter Configuration File 27-1
XParameter Rejection 27-2
Internal XParameters 27-2

Preface

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

Audience

This book is intended for system administrators.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

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

- *Oracle Communications Services Gatekeeper Accounts and SLAs Guide*
- *Oracle Communications Services Gatekeeper Alarm Handling Guide*
- *Oracle Communications Services Gatekeeper Application Developer's Guide*
- *Oracle Communications Services Gatekeeper Communication Service Guide*
- *Oracle Communications Services Gatekeeper Concepts Guide*
- *Oracle Communications Services Gatekeeper Installation Guide*
- *Oracle Communications Services Gatekeeper Java API Reference*
- *Oracle Communications Services Gatekeeper Licensing Guide*
- *Oracle Communications Services Gatekeeper OAM Java API Reference*
- *Oracle Communications Services Gatekeeper OAuth Guide*
- *Oracle Communications Services Gatekeeper Partner Relationship Management Guide*

- *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide*
- *Oracle Communications Services Gatekeeper Platform Test Environment Guide*
- *Oracle Communications Services Gatekeeper Release Notes*
- *Oracle Communications Services Gatekeeper RESTful Application Developer's Guide*
- *Oracle Communications Services Gatekeeper SDK User's Guide*
- *Oracle Communications Services Gatekeeper Security Guide*
- *Oracle Communications Services Gatekeeper System Backup and Restore Guide*

1 Introduction

This guide describes Oracle Communications Services Gatekeeper from operational, management and provisioning perspectives.

Container Services

The following chapters describe management of container services:

- Managing Management Users and Management User Groups
- Managing and Configuring Budgets
- Managing and Configuring EDRs, CDRs and Alarms
- CDRs and Diameter
- Managing and Configuring Statistics and Transaction Licenses
- Setting Up Geographic Redundancy
- Managing and Configuring the SNMP Service
- Managing and Configuring the Trace Service
- Managing and Configuring the Storage Service
- Managing and Configuring Credit Control Interceptors and SLAs
- Securing Web Services and OAM MBeans
- Deployment Model for Communication Services and Container Services
- Hitless Upgrade Using Production Redeployment

Overall Configuration Workflow

This section describes an overall workflow for configuring Oracle Communications Services Gatekeeper.

1. To set up administrative users, see: "Managing Management Users and Management User Groups".
2. To configure Oracle Communications Services Gatekeeper container service, see the following chapters:
   - "Managing and Configuring Budgets"
   - "Managing and Configuring EDRs, CDRs and Alarms"
   - "Managing and Configuring the SNMP Service"

- "Managing and Configuring Statistics and Transaction Licenses"
- "Managing and Configuring the Trace Service"
- "Securing Web Services and OAM MBeans"
- "Managing and Configuring the Plug-in Manager"
- "Managing and Configuring the Tier Routing Manager"
- "Setting Up Geographic Redundancy"

3. To configure Oracle Communications Services Gatekeeper to work with Open Services Architecture (OSA)/Parlay Gateways, see the following chapter:
   - "Managing OSA/Parlay Gateway Connections using Parlay_Access"
4. To configure Security for Web Services and Operation, Administration, and Maintenance (OAM) MBeans, see the following chapter:
   - "Securing Web Services and OAM MBeans" for an introduction and pointers to relevant information.
5. To configure and seethe Plug-in Manager, see the following chapter:
   - "Managing and Configuring the Plug-in Manager"
6. To set up geographically redundant site pairs (as necessary), see the following chapter:
   - "Setting Up Geographic Redundancy".
7. To configure SOA Facades (as necessary), see the following chapter:
   - "Managing and Configuring SOA Facades".

To configure and manage the communication services, see the *Oracle Communications Services Gatekeeper Communication Service Guide*.

After the system is configured, provision service providers and applications, as described in "Creating and Maintaining Service Provider and Application Accounts" in the *Oracle Communications Services Gatekeeper Accounts and SLAs Guide*.

2 Starting and Stopping Servers

This chapter describes how to start and stop servers in an Oracle Communications Services Gatekeeper domain.

Note that the Oracle Communications 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.

Because a typical Oracle Communications Services Gatekeeper domain contains multiple Access Tier and Network Tier servers, with dependencies among the different server types, you should generally adhere to the following sequence when starting up a domain:

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. The Administration Server can also be used to monitor the startup/shutdown status of each Managed Server. You generally 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 servers in each partition.

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

Using Scripts

You can start Network Tier and Access Tier servers by using either the `startManagedWebLogic` script installed with the Configuration wizard or a 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
```

> **Note:** By default, the servers are started in production mode. This means that user credentials must be provided. For more information, see "Provide User Credentials to Start and Stop Server" in *Oracle® Fusion Middleware Node Manager Administrator's Guide for Oracle WebLogic Server* at
>
> http://download.oracle.com/docs/cd/E15523_01/web.1111/e13708/overview.htm

Using Node Manager

You can also start Network Tier and Access Tier servers by using the Administration Console in conjunction with an instance of Node Manager running on each machine. There are many different ways to use Node Manager. See *Oracle Weblogic Middleware Node Manager Administrator's Guide for Oracle WebLogic Server* at

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13740/toc.htm

but the easiest is to use the Java-based version, as follows:

> **Note:** The following instructions assume UNIX or Linux. Equivalent Windows versions exist, but Windows is not supported for production servers. The following instructions must be followed on managed server.

1. 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_10.3/server/bin** directory.

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

2. Add the domains that the Node Manager instance controls to the *Middleware_Home***/wlserver_10.3/common/nodemanager/nodemanager.domains** file. See “General Node Manager Configuration” in *Oracle® Fusion Middleware Node Manager Administrator's Guide for Oracle WebLogic Server* at

   http://download.oracle.com/docs/cd/E15523_01/web.1111/e13740/nodemgr_config.htm#BHCJHHDC

   for a description of **nodemanager.domains**. A sample entry for a domain in this file is:

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

3. Edit the *Middleware_Home***/wlserver_10.3/common/nodemanager/nodemanager.properties** file. Make sure that `StartScriptEnabled=true` is set.

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

5. Create a startup script. In the *Domain_home* directory, create a file called **startWeblogic.sh** and add this line:

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

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

6. Make sure that Listen Address is configured in the Console. In **Domain Structure**, click **Environment** > **Machines** > *machine_name* > **Node Manager.** You must use **Lock & Edit** to make any changes.
7. After everything is set up, to use the Node Manager to start Oracle Communications Services Gatekeeper servers, go to the domain's Administration Console. Under **Environment**, select the managed servers you want to start.

For other ways of starting and stopping servers, see "Starting and Stopping Servers" in *Oracle® Fusion Middleware Managing Server Startup and Shutdown for Oracle WebLogic Server* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13708/toc.htm

HP-UX

If you are using HP-UX in your installation, you must make a small edit to whichever of the startup scripts you are using. You must add the `-Djava.security.egd` flag to the invocation, in the locations shown below in bold.

```
if [ '${WLS_REDIRECT_LOG}' = '' ] ; then
        echo 'Starting WLS with line:'
        echo '${JAVA_HOME}/bin/java ${JAVA_VM} ${MEM_ARGS} ${JAVA_
OPTIONS}-Dweblogic.Name=${SERVER_NAME} -Djava.security.policy=${WL_
HOME}/server/lib/weblogic.policy  ${PROXY_SETTINGS}
${SERVER_CLASS}' ${JAVA_HOME}/bin/java ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}
-Djava.security.egd=/dev/random -Dweblogic.Name
=${SERVER_NAME} -Djava.security.policy=${WL_HOME}/server/lib/weblogic.policy
${PROXY_SETTINGS} ${SERVER_CLASS}
else
        echo 'Redirecting output from WLS window to ${WLS_REDIRECT_LOG}'
        ${JAVA_HOME}/bin/java ${JAVA_VM} ${MEM_ARGS} ${JAVA_OPTIONS}
-Djava.security.egd=/dev/random -Dweblogic.Name
=${SERVER_NAME} -Djava.security.policy=${
WL_HOME}/server/lib/weblogic.policy${PROXY_SETTINGS} ${SERVER_CLASS}  >'${WLS_
REDIRECT_LOG}' 2>&1
```

Solaris 64-bit

If your installation is for 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.

3 Operation and Maintenance: General

This chapter gives an overview of the mechanisms available for controlling Oracle Communications Services Gatekeeper.

Administration Console Overview

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

At least one Network Tier server must be started before you log in to the WebLogic Server Administration Console. Services Gatekeeper servers started after you log in to the Administration Console are not displayed in the Administration Console. To see them, you must log in again.

Starting and Using the Administration Console

Start the Administration Console by entering this URL in a supported Web browser:

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. **localhost** works if you are running the Administration Console on the same system as the Administration Server.

port is the port to connect to. The default is **8001**.

Log with your credentials.

> **Note:** If this is the first time Services Gatekeeper is started, use the username `weblogic` and password `weblogic`, which is the recommended value for the installation and initial domain configuration stage.After you have logged in, create an administrative user using the instructions in "Managing Management Users and Management User Groups", and remove the user `weblogic`. Either remove or change password for the WebLogic Server user, see the on-screen help text on the Administration Console user interface.

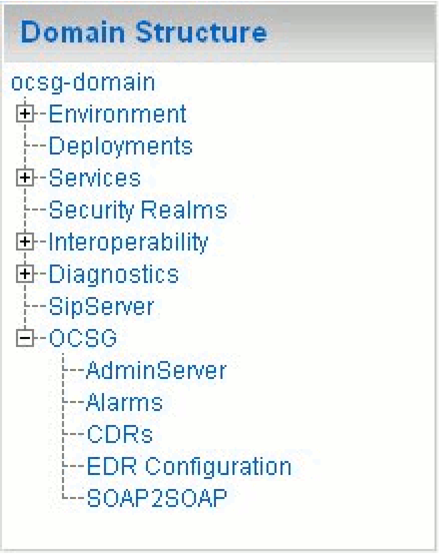
All Services Gatekeeper configuration and monitoring is provided through the following nodes in the left pane of the console, in the **Domain Structure** group:

- OCSG - Container for all Services Gatekeeper servers.

- <Server Name> - One entry per Services Gatekeeper server.
  - <Server Name> - Clicking on this link displays the MBeans pane, which displays all OAM objects available for the selected Services Gatekeeper server. Some configuration settings are cluster-wide, and other settings are per server.
  - Alarms - clicking this link displays the WebLogic Oracle Communications Services Gatekeeper Alarms Pane.
  - CDRs - Clicking this link displays the Oracle Communications Services Gatekeeper CDRs Pane.
  - EDR Configuration - Clicking this link displays the EDR Configuration pane; see the *Oracle Communications Services Gatekeeper System Administration Guide* section "Managing EDR, CDR, and alarms configuration files using the EDR Configuration Pane".
- SipServer - Clicking this link displays the Oracle Communications Converged Application Server's administration console, see the *Oracle Communications Services Gatekeeper System Administration Guide* section "Converged Application Server Administration Console for SIP-based Services".

Use the WebLogic Server administration console to configure SIP settings. See *Oracle Communications Converged Application Server* documentation.

Figure 3–1 Domain Structure – Links to Administration Console for Oracle Communications Services Gatekeeper

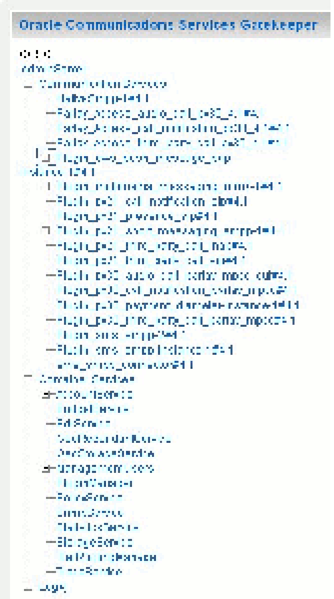


MBeans pane

The MBeans pane contains a tree structure of all management objects applicable for the selected Oracle Communications Services Gatekeeper server.

By clicking on a management object, the corresponding Configuration and Provisioning page for the management object is displayed immediately below the MBean pane: see the *Oracle Communications Services Gatekeeper System Administration Guide* section "Configuration and Provisioning".

Figure 3–2 MBeans Pane



Oracle Communications Services Gatekeeper

Configuration and Provisioning

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.

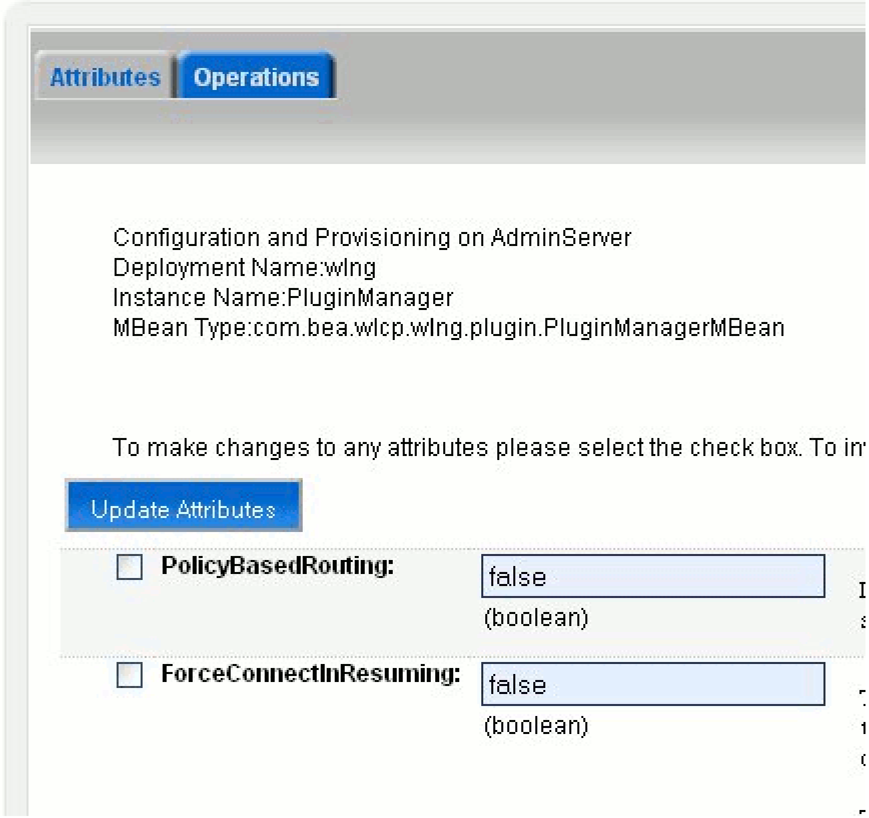
> **Note:** In this document, read-only attributes are indicated by **(r)** after the name.

Read-write attributes have a check-box next to them.

To change an attribute:

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

Figure 3–3 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 operations for the managed object.

Operations either display data, set data, or perform an actual task.

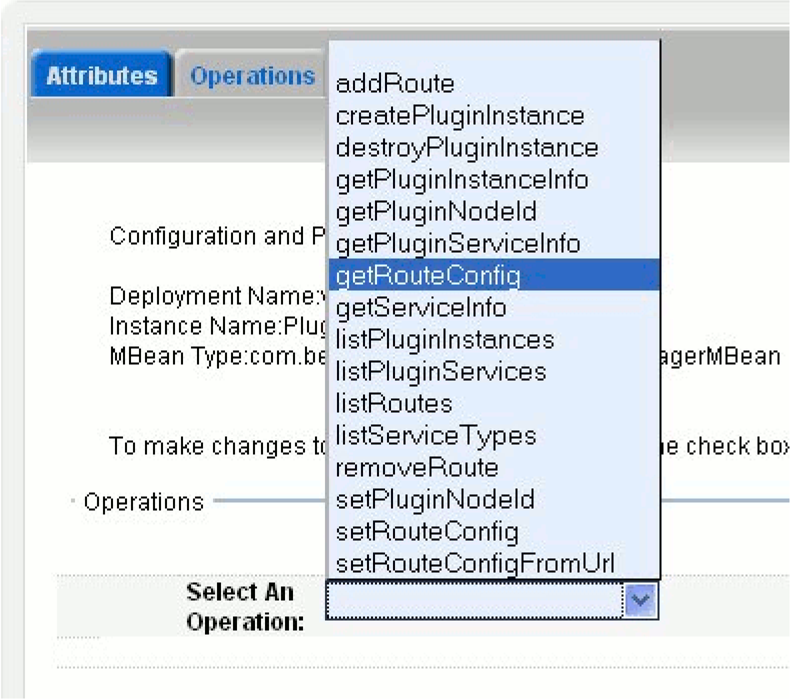
To perform an operation:

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

   The fields for the operation are displayed.

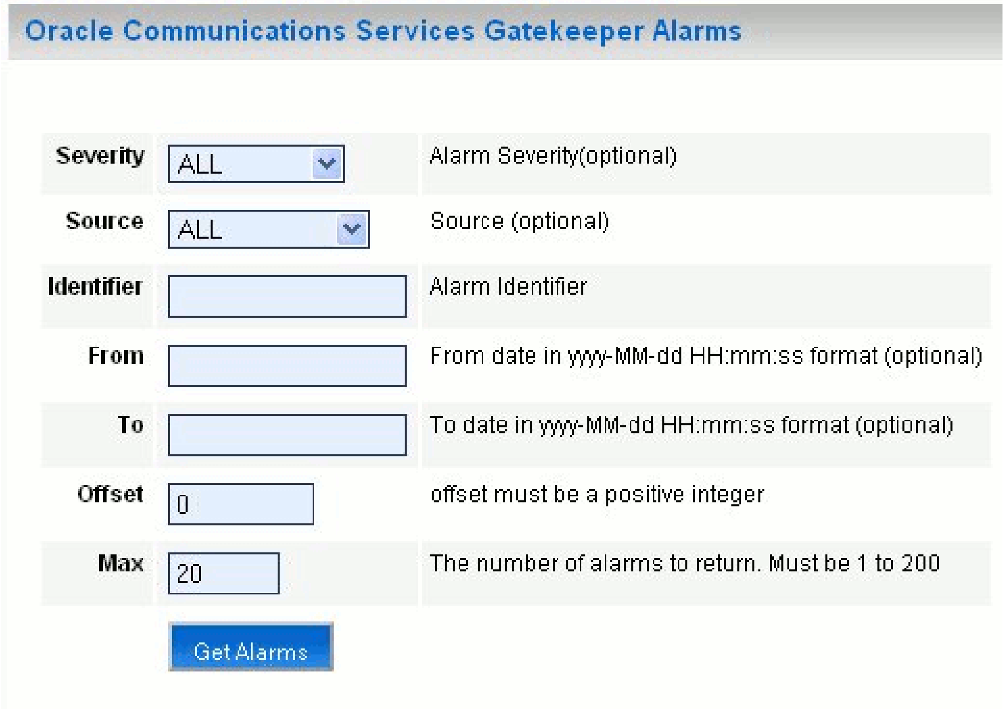
2. Enter the information in the fields.
3. Click **Invoke**.

Figure 3–4 Example Configuration and Provisioning Pane Operations Tab



WebLogic Oracle Communications Services Gatekeeper Alarms Pane

Figure 3–5 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 Oracle Communications Services Gatekeeper Alarms pane displays alarms emitted.

It is possible to filter the output to the page on a set of criteria; see Table 3–1.

The list is returned by clicking **Get Alarms**.

Table 3–1 Oracle Communications Services Gatekeeper Alarms Pane

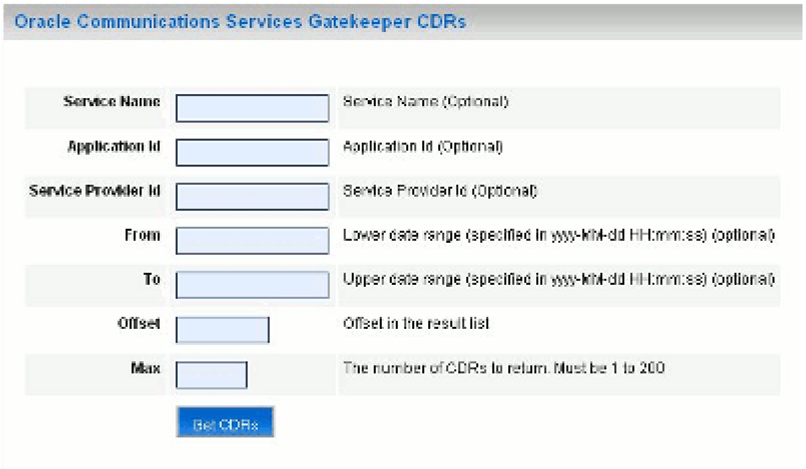
Filter on...	Input
Severity	From the **Severity** list, select which severity level to display. The options are: - **ALL** - **WARNING** - **MINOR** - **MAJOR** - **CRITICAL**
Source	From the **Server** list, select the server from which to display alarms. The options are: - **ALL**, for all servers - **<Server name>**, for an individual server
Identifier.	See *Oracle Communications Services Gatekeeper Alarm 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 3–1 (Cont.) Oracle Communications Services Gatekeeper Alarms Pane

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.

Oracle Communications Services Gatekeeper CDRs Pane

Figure 3–6 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

The Services Gatekeeper Charging Data Records (CDRs) pane displays CDRs that have been generated.

It is possible to filter the output to the page on a set of criteria, see Table 3–2.

The list is returned by clicking **Get CDRs**.

Table 3–2 Oracle Communications Services Gatekeeper CDRs Pane

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 3–2 (Cont.) Oracle Communications Services Gatekeeper CDRs Pane

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

There is an administration console for the Oracle Communications Converged Application Server. This handles setting for Converged Application Server and the SIP Servlet parts of Services Gatekeeper communication services.

Click *Domain Name* > SipServer to open the Converged Application Server administration console.

Java Management Extensions

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 OAM can be found in the Reference topic page in the Oracle Communications 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.

Following is an example of connecting to the MBean server and performing an operation on the JMX interfaces.

The MBean object name is versioned. The MBean name includes the version number for the release. External JMX clients need to be updated according to the release number.

Example 3–1 Example of using JMX to manage Oracle Communications 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 setting an attribute or calling an operation fails, a `com.bea.wlcp.wlng.api.management.ManagementException`, or a subclass of this exception is thrown. The following subclasses are defined:

`DuplicateKeyException` – Thrown when the operation creates a duplicate key.

`InputManagementException` – Thrown for invalid user input.

`KeyNotFoundException` – Thrown when the operation cannot find the specified key.

A JMX client must have the same version of exception classes available; otherwise the client will throw `ClassNotFoundException`.

Middleware_Home/**ocsg_pds_5.1/lib/wlng/oam.jar** contains custom classes for OAM and return types.

Middleware_Home/**ocsg_pds_5.1/doc/javadoc_oam** contains Javadoc for OAM.

WebLogic Scripting Tool (WSLT)

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 "Using the WebLogic Scripting Tool" in *Oracle® Fusion Middleware Oracle WebLogic Scripting Tool* at:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13715/using_wlst.htm`

The following topics are covered in this section of the *Oracle Communications Services Gatekeeper System Administration Guide*:

- Working in Interactive Mode
  - Starting WLST and connecting to Services Gatekeeper
  - Exiting WLST
  - Changing an attribute
  - Invoking an operation
- WebLogic Scripting Tool (WSLT)

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

Working in Interactive Mode

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

Starting WLST and connecting to Services Gatekeeper

1. Make sure the correct Java environment is set:

   UNIX: *Domain_Home*`/bin/setDomainEnv.sh`

   Windows: *Domain_Home*`\bin\setDomainEnv.cmd`

2. Start WLST:

   ```
   java weblogic.WLST
   ```

3. Connect to the server to manage:

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

4. Change to the custom tree where Oracle Communications Services Gatekeeper MBeans are located:

   ```
   custom()
   ```

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

5. Display a list of the Mbeans:

   ```
   ls()
   ```

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

6. Select the MBean to change:

   ```
   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')
   ```

Exiting WLST

1. Disconnect from Oracle Communications Services Gatekeeper:

   ```
   disconnect()
   ```

2. Exit the WLST shell:

   ```
   exit()
   ```

Changing an attribute

1. Select the MBean to change attribute for.

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: `set('BatchSize', 2001)`

The attribute values for an MBean can be displayed using `ls()`.

Invoking an operation

1. Select the MBean to invoke an operation on.
2. Define the parameters as an array.
3. Define the data types as an array.
4. Invoke the operation.

For example, to invoke the `displayStatistics` method in the EDRService MBean:

```
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))
```

This operation has no arguments.

For example, to add a route using the MBean for the Plug-in Manager:

```
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)
```

The method `addRoute` takes two arguments, and the values of the parameters are stated in the same order as the parameters are defined in the signature of the method.

> **Note:** For a method that has input parameters and a string return value, the `invoke` command can be executed as follows:

```
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

When using WLST together with scripts, WLST is invoked as follows:

`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. It may be useful to specify login information and connection information as arguments to the scripts.

The following script connects to Oracle Communications Services Gatekeeper and changes to an MBean defined by argument.

Example 3–2 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)
```

For example, to invoke the script:

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

Database Maintenance

This section lists the three database tables that you should clean periodically and provides and example script for doing so.

About Cleaning Database Tables

Table 3–3 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 3–3 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 the *Oracle Communications Services Gatekeeper System Administrator's Guide* section "About Cleaning EDR Analytics Tables" for details.
EDR Analytics	TRANSACTION_PARAMETER_HISTORY	Daily if used. See the *Oracle Communications Services Gatekeeper System Administrator's Guide* section "About Cleaning EDR Analytics Tables" for details.
EDR Analytics	TRANSACTION_INVALID_HISTORY	Daily if used. See the *Oracle Communications Services Gatekeeper System Administrator's Guide* section "About Cleaning EDR Analytics Tables" for details.

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

Type of Database Table	Table	Recommended Cleaning Interval
EDR Analytics	TRANS_PARA_INVALID_ HISTORY	Daily if used. See the *Oracle Communications Services Gatekeeper System Administrator's Guide* section "About Cleaning EDR Analytics Tables" for details.

About Cleaning EDR Analytics Tables

You are only required to clean the EDR analytics tables listed in Table 3–3 if you use the EDR analytics feature and have **EdrAanalyticMBean.StoreHistory** is set true. The cleaning interval depends on the amount of traffic stored in your database, but these tables usually 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.

Typically 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 charging data records (CDRs), which are a type of event data record (EDR). Typically CDRs are integrated by means of EDR listeners. You can also listen for alarms by setting up an SNMP/EDR listener. If you do this, you could disable CDR and alarm storage in the database by setting the EDR Service's StoreAlarms or Store CDRs attributes to `false`. However, Oracle recommends storing alarms for some period of time, for trouble-shooting purposes. See the *Oracle Communications Services Gatekeeper System Administrator's Guide* sections Attribute: StoreCDRs and Attribute: StoreAlarms.

For general information about EDRs, CDRs, and alarms see Oracle Communications Services Gatekeeper System Administrator's Guide Chapter 8, "Managing and Configuring EDRs, CDRs and Alarms." For more detailed information about creating an EDR listener see "Creating an EDR Listener and Generating SNMP MIBs" in the *Platform Development Studio Developer's Guide*.

For information about setting the expiration of data stored in the database tables, see the Oracle Communications Services Gatekeeper System Administrator's Guide section Storage Data Expiration.

If your Services Gatekeeper implementation has extremely 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 ever 2-3 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 may not require any server restart.

Creating a Script to Clean SLEE_CHARGING

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

1. Create a table called SLEE_CHARGING_BACKUP to use as a repository for SLEE_CHARGING data.
2. Create an SQL query that:
   a. Selects all data in SLEE_CHARGING 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. The exact timing is up to you.

   b. Insert the selected data/rows into SLEE_CHARGING_BACKUP.

   c. Delete the same data from original 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~
```

Create a similar statement like the one below to archive the data into a new table called SLEE_STATISTICS_DATA:

```
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;
```

4

Managing Management Users and Management User Groups

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

Task Overview

Management of Services Gatekeeper is performed by administrative users. There are a set of management users, identified by their *user type*. Each management user is also assigned a *user level*.

Table 4–1 provides an overview of the operations for managing management users.

Table 4–1 Operations Associated with Management Tasks

To...	Use
Create an administrative user	Operation: addUser
Change password	Operation: changeUserPassword
Delete an administrative user	Operation: deleteUser
Get user level	Operation: getUserLevel
List administrative users	Operation: listUsers

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.
- Management users are users who have access to and can perform management and administration functions.

Traffic users cannot login to the Administration Console or perform any management operations.

During installation, default groups are created in the WebLogic Server Embedded LDAP server. Table 4–2 lists the names of the default user groups, their membership criteria, and classification of user roles.

Table 4–2 User Groups and Privileges

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

When an Application Instance sends a Simple Object Access Protocol (SOAP) request to the application-facing interfaces, it is authenticated by the *WLNG Application Authenticator*; upon successful authentication, it adds the WLNGTrafficUsers group to the user principals, in addition to the service provider ID, application ID, service provider group ID, and application group ID.

When Management users log in successfully, they are added to the oamUser group.

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*.

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

A policy contains one or more conditions. For example, a simple policy can be *Allow access if the user belongs to Admin Role*.

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.

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

User Level

Management users are assigned different user levels based on which JMX resources they will be able to access. Table 4–3 lists the access privileges associated with user levels on Services Gatekeeper and WebLogic Server.

Table 4–3 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

At a more granular level, an administrator may want to give access to only a subset of management interfaces. This can be achieved by applying XACML policies.

Following is an outline of how 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 role to the group
5. Add an XACML policy to the user group. It is possible to restrict access at a granular level; MBean, MBean attribute, or MBean operation level. See Understanding WebLogic Resource Security in *Oracle WebLogic Server Securing WebLogic Resources Using Roles and Policies* at

   `http://docs.oracle.com/cd/E24329_01/web.1211/e24421/understdg.htm#i121442996`

   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*.

Reference: Attributes and Operations for ManagementUsers

Managed object: Container Services->ManagementUsers->ManagementUsers

MBean: com.bea.wlcp.wlng.user.management.ManagementUserMBean

Following is a list of attributes and operations for configuration and maintenance:

- Operation: addUser
- Operation: changeUserPassword
- Operation: deleteUser
- Operation: getUserLevel
- Operation: listUsers

Operation: addUser

Scope: Cluster

Adds a Services Gatekeeper administrative user.

Signature:

```
addUser(Username:String, Password: String, userLevel: int, type: int)
```

Table 4–4 describes these parameters.

Table 4–4 Parameters for addUser

Parameter	Description
Username	User name
Password	Password
UserLevel	Defines the user level when administrating Oracle Communications Services Gatekeeper. See "User Level".
Type	Type of management user. Use: ■ 0 for management user ■ 1 for PRM operator user ■ 2 for PRM service provider user See "User Types".

Operation: changeUserPassword

Scope: Cluster

Changes the password for an existing Services Gatekeeper administrative user.

Signature:

```
changeUserPassword(UserName: String, OldPasswd: String, NewPasswd: String)
```

Table 4–5 describes these parameters.

Table 4–5 Parameters for changeUserPassword

Parameter	Description
UserName	User ID for administrative user
OldPasswd	Current password
NewPasswd	New password

Operation: deleteUser

Scope: Cluster

Deletes an Services Gatekeeper administrative user.

Signature:

```
deleteUser(UserName: String)
```

Table 4–6 describes this parameter.

Table 4–6 Parameters for deleteUser

Parameter	Description
UserName	User ID for administrative user

Operation: getUserLevel

Scope: Cluster

Gets the user level for a management user. See "User Level".

Signature:

```
getUserLevel(UserName: String)
```

Table 4–7 describes this parameter.

Table 4–7 Parameters for getUserLevel

Parameter	Description
UserName	User ID for the management user

Operation: listUsers

Scope: Cluster

Displays a list of all registered management users and their corresponding user levels. See "User Level".

```
listUsers(Type: int, Offset: int, Size: int)
```

Table 4–8 describes these parameters.

Table 4–8 Parameters for listUsers

Parameter	Description
Type	Type of user. Use: ■ 0 for management user ■ 1 for PRM operator user ■ 2 for PRM service provider user See "User Types".
Offset	Offset in the list. Starts with 0.
Size	Size of the list

Reference: Attributes and Operations for ManagementUserGroup

Managed object: Container Services->ManagementUsers->ManagementUserGroup

MBean: com.bea.wlcp.wlng.user.management.ManagementUserGroupMBean

Following is a list of attributes and operations for configuration and maintenance:

- Operation: addUsertoGroup
- Operation: createUserGroup
- Operation: listGroups
- Operation: listUsers

Operation: addUsertoGroup

Scope: Cluster

Adds an Oracle Communications Services Gatekeeper administrative user to a user group.

Signature:

```
addUsertoGroup(Username:String, GroupName: String)
```

Table 4–9 describes these parameters.

Table 4–9 Parameters for addUsertoGroup

Parameter	Description
Username	User name
GroupName	Group name

Operation: createUserGroup

Scope: Cluster

Creates a new user group.

Signature:

```
createUserGroup(GroupName: String, Description: String)
```

Table 4–10 describes these parameters.

Table 4–10 Parameters for createUserGroup

Parameter	Description
GroupName	Name of the new administrative group
Description	A textual description

Operation: listGroups

Scope: Cluster

Lists all registered user groups.

Signature:

```
listGroups(Offset: int, Size: int)
```

Table 4–11 describes these parameters.

Table 4–11 Parameters for listGroups

Parameter	Description
Offset	Offset in the list. Starts with 0.
Size	Size of the list

Operation: listUsers

Scope: Cluster

Lists user based on user group.

Signature:

```
listUsers(GroupName: String, Offset: int, Size: int)
```

Table 4–12 describes these parameters.

Table 4–12 Parameters for listUsers

Parameter	Description
GroupName	Group name
Offset	Offset in the list. Starts with 0.
Size	Size of the list

5

Managing and Configuring SOA Facades

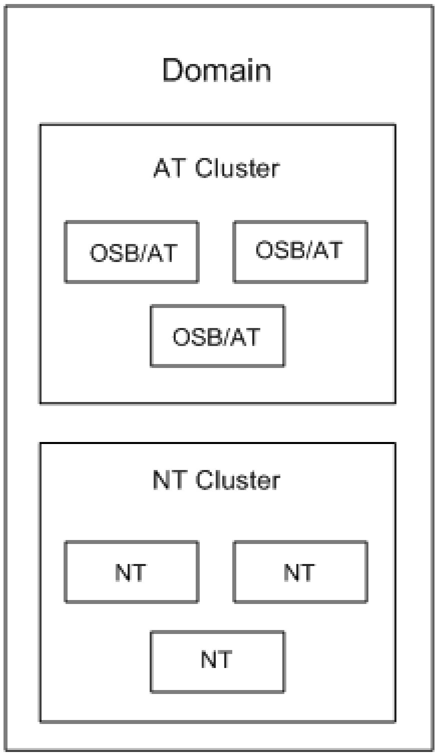
This chapter describes how to configure the Oracle Communications Services Gatekeeper Service Oriented Architecture (SOA) Facades in Oracle Service Bus (OSB).

Introduction

The Oracle SOA Suite is an Oracle Fusion Middleware Component that allows you to assemble multiple services into SOA composite applications for service consumers to use. You administer the SOA Suite using the Oracle Enterprise Manager Fusion Middleware Control Console. The SOA Suite relies on another Oracle Middleware Component, the Oracle Service Bus (OSB). It is the OSB that you must install and configure with Service Gatekeeper so that it communicates with the SOA Facades.

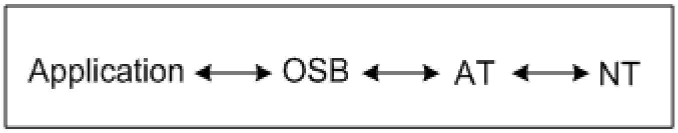
To provide SOA support, Services Gatekeeper domains must have each AT server collocated with OSB. Figure 5–1 shows a Services Gatekeeper domain with the OSB and AT collocated on the same server.

Figure 5–1 Services Gatekeeper Domain with SOA Support



The OSB then interfaces with the Services Gatekeeper NT layer. Figure 5–2 shows how an application request flows through the OSB to the AT tier and finally to the NT tier. Return communication flows back in reverse order.

Figure 5–2 Services Gatekeeper-SOA Communication Order



During installation, you configure OSB using the SOA domain configuration template provided with Services Gatekeeper, and manage SOA Facades using the OSB Administration Console.

The OSB administration console is located at `http://`*`admin_server_IP_addr`*`/sbconsole`. For each communication service there is a set of business services and proxy services, one for each SOAP interface.

You do not need to configure the SOA Facades, but both the business services and the proxy services are configured in the same manner as other OSB projects. You load SOA Facades projects using the Service Bus Console.

Upgrading SOA Facade Projects From Previous Release

You need to upgrade any existing SOA facade projects created in previous releases of Service Gatekeeper to this release. To do this you import these SOA facade projects into Services Gatekeeper using System Administration console, and then export them again using these steps:

1. (Optional) Save copies of the old SOA facade projects .jar files a neutral location.
2. Follows the steps in "Loading SOA Facade Projects" to load your existing projects into Services Gatekeeper.
3. Under **System Administration**, click on **Export Resources**.
4. Select all SOA projects.
5. Click **Export**.

   Export the files to their original locations, overwriting the original files with the updated versions.

You can now use your SOA facade projects with Services Gatekeeper 5.1.

Loading SOA Facade Projects

This section describes how to load the SOA Facade projects.

The SOA Service Facades are not loaded by default. Load the projects for the Communication Services that are to be used using the instruction below.

See "Available SOA Facades" for a list of available SOA Facades and the deployment artifacts for them.

1. In the Service Bus administration console, click on **System Administration**.
2. Under **System Administration**, click on **Import Resources**.
3. Click the **Browse** button next to the **File Name** entry field.

4. Browse to the directory *domain_home*/`soa` and select the .jar file, according to Table 5–1.
5. In the **Import Resources- Project JAR File** screen, click the **Import** button.

> **Note:** In the **Change Center**, click the **Activate** button.

Available SOA Facades

Table 5–1 lists the SOA Facades for Services Gatekeeper. The files are located in *domain_home*/`soa`.

Table 5–1 SOA Facade projects and services

Service Facade	Project	File
Not applicable	soa_common	File: sb_common.jar Business Service: Not applicable Proxy Service: Not applicable Type: Use for both application-initiated and network-triggered requests.
Session Manager	soa_session	File: sb_session.jar Business Service: SessionManager_BS Proxy Service: SessionManager_PS Type: Use for application-initiated requests.
Parlay X 2.1 Audio Call	soa_audio_call_pz21	File: sb_ac_px21.jar Business Service: AudioCall_BS Proxy Service: AudioCall_PS Type: Use for application-initiated requests.
Parlay X 3.0 Audio Call	soa_audio_call_px30	File: sb_ac_px30.jar Business Service: AudioCallCaptureMedia_BS Business Service: AudioCallPlayMedia_BS Proxy Service: AudioCallCaptureMedia_PS Proxy Service: AudioCallPlayMedia_PS Type: Use for application-initiated requests.

Table 5–1 (Cont.) SOA Facade projects and services

Service Facade	Project	File
Parlay X 2.1 Call Notification	soa_call_notification_px21	File: sb_cn_px21.jar Business Service: CallDirection_BS Business Service: CallDirectionManager_BS Business Service: CallNotification_BS Business Service: CallNotificationManager_BS Proxy Service: CallDirection_PS Proxy Service: CallDirectionManager_PS Proxy Service: CallNotification_PS Proxy Service: CallNotificationManager_PS Type: Use for network-triggered requests.
Parlay X 3.0 Call Notification	soa_call_notification_px30	File: sb_cn_px30.jar Business Service: CallDirection_BS Business Service: CallDirectionManager_BS Business Service: CallNotification_BS Business Service: CallNotificationManager_BS Proxy Service: CallDirection_PS Proxy Service: CallDirectionManager_PS Proxy Service: CallNotification_PS Proxy Service: CallNotificationManager_PS Type: Use for network-triggered requests.
Parlay X 2.1 Multimedia Messaging	soa_mms_px21	File: sb_mms_px21.jar Business Service: MessageNotification_BS Business Service: MessageNotificationManager_BS Business Service: ReceiveMessage_BS Business Service: SendMessage_BS Proxy Service: MessageNotification_PS Proxy Service: MessageNotificationManager_PS Proxy Service: ReceiveMessage_PS Proxy Service: SendMessage_PS Type: Use for both application-initiated and network-triggered requests.

Table 5–1 (Cont.) SOA Facade projects and services

Service Facade	Project	File
Parlay X 3.0 Payment	soa_payment_px30	File: sb_payment_px30.jar Business Service: AmountCharging_BS Business Service: ReserveAmountCharging_BS Proxy Service: AmountCharging_PS Proxy Service: ReserveAmountCharging_PS Type: Use for application-initiated requests.
Parlay X 2.1 Presence	soa_presence_px21	File: sb_presence_px21.jar Business Service: PresenceConsumer_BS Business Service: PresenceNotification_BS Business Service: PresenceSupplier_BS Proxy Service: PresenceConsumer_PS Proxy Service: PresenceNotification_PS Proxy Service: PresenceSupplier_PS Type: Use for both application-initiated and network-triggered requests.
Parlay X 2.1 Short Messaging	soa_sms_px21	File: sb_sms_px21.jar Business Service: ReceiveSms_BS Business Service: SendSms_BS Business Service: SmsNotification_BS Business Service: SmsNotificationManager_BS Proxy Service: ReceiveSms_PS Proxy Service: SendSms_PS Proxy Service: SmsNotification_PS Proxy Service: SmsNotificationManager_PS Type: Use for both application-initiated and network-triggered requests.
Extended Web Services Subscriber Profile	soa_subscriber_ews	File: sb_subscriber_ews.jar Business Service: SubscriberProfile_BS Proxy Service: SubscriberProfile_PS Type: Use for application-initiated requests.
Parlay X 2.1 Third Party Call	soa_third_party_call_px21	File: sb_tpc_px21.jar Business Service: ThirdPartyCall_BS Proxy Service: ThirdPartyCall_PS Type: Use for application-initiated requests.

Table 5–1 (Cont.) SOA Facade projects and services

Service Facade	Project	File
Parlay X 3.0 Third Party Call	soa_third_party_call_px30	File: sb_tpc_px30.jar Business Service: ThirdPartyCall_BS Proxy Service: ThirdPartyCall_PS Type: Use for application-initiated requests.
Parlay X 2.1 Terminal Status	soa_ts_px21	File: sb_ts_px21.jar Business Service: TerminalStatusNotificationManager_BS Proxy Service: TerminalStatusNotificationManager_PS Type: Use for both application-initiated and network-triggered requests.
Parlay X 2.1 Terminal Location	soa_tl_px21	File: sb_tl_px21.jar Business Service: TerminalLocation_BS Business Service: TerminalLocationNotification_BS Business Service: TerminalLocationNotificationManager_BS Proxy Service: TerminalLocation_PS Proxy Service: TerminalLocationNotification_PS Proxy Service: TerminalLocationNotificationManager_PS Type: Use for both application-initiated and network-triggered requests.
Parlay X 3.0 Device Capabilities and Configuration	soa_device_capabilities_px30	File: sb_dc_px_30.jar Business Service: DeviceCapabilities_BS Proxy Service: DeviceCapabilities_PS Type: Use for application-initiated requests.
Extended Web Services WAP Push	soa_wap_ews	File: sb_wap_ews.jar Business Service: PushMessage_BS Business Service: PushMessageNotification_BS Proxy Service: PushMessage_PS Proxy Service: PushMessageNotification_PS Type: Use for application-initiated requests.

Table 5–1 (Cont.) SOA Facade projects and services

Service Facade	Project	File
Extended Web Services Binary SMS	soa_sms_ews	File: sb_sms_ews.jar Business Service: BinarySms_BS Business Service: BinarySmsNotification_BS Business Service: BinarySmsNotificationManager_BS Proxy Service: BinarySms_PS Proxy Service: BinarySmsNotification_PS Proxy Service: BinarySmsNotificationManager_PS Type: Use for both application-initiated and network-triggered requests.

6

Configuring Coherence

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

About Configuring 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. In a clustered environment you need to create a Coherence configuration.

You can declare the Coherence cluster-related attributes like multicast address and port, unicast address and port overrides using the administration console. WebLogic stores the attributes in the WebLogic configuration repository and makes them available to the Coherence cluster service.

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.

For details about Coherence configuration, see section “Operational Configuration Elements” in *Oracle Communications Services Gatekeeper Coherence Developer’s Guide*.

Configuring Coherence

To configure Coherence:

1. Log in to the administration console.
2. Expand the **Services** section in the **Domain Structure** pane.
3. Click on **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. Make sure that **Use a custom Cluster Configuration File** is not selected.
7. Click **Next**.

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

8. If you are using unicast addressing:
   a. Enter the IP address for the cluster unicast listener in the **Unicast Listen Address** field.
   b. Enter the port for the cluster unicast listener in the **Unicast Listen Port** field.
   c. Check **Unicast Port Auto Adjust** if the unicast port number should be automatically incremented if the port cannot be bound because it is already in use.
9. If you are using multicast addressing:
   a. Enter the IP address for the cluster multicast listener in the **Multicast Listen Address** field.
   b. Enter the port for the cluster multicast listener in the **Multicast Listen Port** field.
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**.

7 Managing and Configuring Budgets

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

Introduction

In Oracle Communications 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 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 Oracle Communications Services Gatekeeper server updates both its own local traffic count and the cluster-wide count maintained in one Oracle Communications 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 Oracle Communications 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: see "Attribute: PersistentBudgetFlushInterval" and "Attribute: PersistentBudgetTimeThreshold".

A desired accuracy factor for synchronizations can also be configured, see "Attribute: AccuracyFactor".

The configuration settings for these attributes affect accuracy and performance:

- The higher the "Attribute: AccuracyFactor", 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 "Attribute: PersistentBudgetFlushInterval", 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 "Attribute: PersistentBudgetTimeThreshold", 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.

Synchronization of budgets

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; see "Attribute: AccuracyFactor".
- **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.

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.

Master Internal

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.

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. See "Attribute: PersistentBudgetTimeThreshold".

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.

Configuration and Management

Configure the following attributes:

- Attribute: PersistentBudgetFlushInterval
- Attribute: PersistentBudgetTimeThreshold
- Attribute: AccuracyFactor
- Attribute: ConfigUpdateInterval

No management operations are available.

Reference: Attributes and Operations for BudgetService

Managed object: Container Services->BudgetService

MBean:
com.bea.wlcp.wlng.core.budget.management.configuration.BudgetServiceMBean

Following is a list of attributes for configuration and maintenance:

- Attribute: PersistentBudgetFlushInterval
- Attribute: PersistentBudgetTimeThreshold
- Attribute: AccuracyFactor
- Attribute: ConfigUpdateInterval

Attribute: PersistentBudgetFlushInterval

Scope: Cluster

Format: int

Units: milliseconds

Specifies the time interval between flushes of budgets to persistent storage. See "Introduction".

Attribute: PersistentBudgetTimeThreshold

Scope: Cluster

Format: int

Units: milliseconds

Specifies threshold value for budgets. Budgets for all time intervals defined in the SLA larger than this value are persisted. See "Introduction".

Attribute: AccuracyFactor

Scope: Cluster

Format: int

Specifies the accuracy factor. See "Introduction".

Attribute: ConfigUpdateInterval

Scope: Cluster

Format: int

milliseconds

Configuration synchronization interval between the slave nodes and the master node.

Adding a Datasource

Under normal operating conditions, Oracle Communications 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 the Oracle Communications Services Gatekeeper at large. A datasource is an abstraction that handles connections with the persistent store. 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”.

> **Note:** Datasource issues are not the only reason this alarm might be thrown.

If you encounter this problem, you can set up a separate singleton datasource for the migration mechanism that will assure that the singleton service always has access to the persistent store. This datasource should be configured to use the same database as the common transactional (XA) datasource (wlng.datasource). For more information on automatic migration of singleton services, see "Service Migration" in *Oracle® Fusion Middleware Using Clusters for Oracle WebLogic Server* at:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13709/toc.htm`

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 for Oracle WebLogic Server* at:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13737/jdbc_datasources.htm`

8

Managing and Configuring EDRs, CDRs and Alarms

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

About EDRs, CDRs, and Alarms

Event Data Records (EDRs), are generated in the following ways:

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

EDR categories and XML markup

EDRs are the base component of both CDRs and alarms.

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**

The configuration file contains a set of sections:

- `<edr-config>` contains descriptors that describe pure EDRs.
- `<alarm-config>` contains descriptors that describe EDRs that should be considered alarms.
- `<cdr-config>` contains descriptors that describe EDRs that should be considered CDRs.

The default Oracle Communications Services Gatekeeper installation comes with a set of predefined descriptors. Changing and adapting the descriptors is done as a part of an integration project.

The XML configuration file can be edited and reloaded using the EDR Configuration pane: see "Managing EDR, CDR, and alarms configuration files using the EDR Configuration Pane".

Example 8–1 illustrates the structure of **wlng-edr.xml**. See “Creating an EDR Listener and Generating SNMP MIBs” in *Oracle Communications Services Gatekeeper Platform Development Studio Developer’s Guide* for more information.

Example 8–1 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>
```

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 Oracle Communications Services Gatekeeper to the network or from the network to Oracle Communications 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 8–1 Application Tier and Network Tier EDR Generation

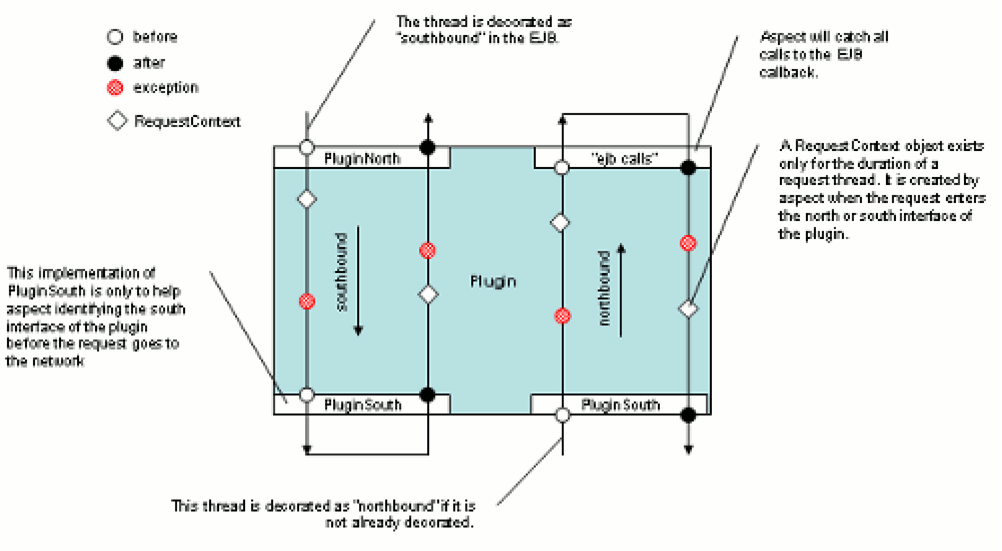

○ before
● after
● exception
◇ RequestContext
The thread is decorated as "southbound" in the EJB.
Aspect will catch all calls to the EJB callback.
A Request Context object exists only for the duration of a request thread. It is created by aspect when the request enters the north or south interface of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

Table 8–1 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 8–1 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
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. May contain multiple addresses.
<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 request method. For example "POST", or "GET".
RequestContext	Attributes in the request context map. (Name/Value pairs)
InterceptorChain	List of all of the interceptors that are triggered
SubscriberID	Subscriber identifier (using route address)

EDRs

All EDRs are passed through the EDRService. All EDRs are dispatched to a JMS-distributed topic so external clients can receive them over JMS.

EDRs are *not* persisted in the database.

Alarms

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

Alarms can be configured to be persisted, see "Alarms".

CDRs

CDRs are EDRs that are mapped to CDRs using the wlng-edr.xml configuration file, see "EDR categories and XML markup".

CDRs can be configured to be persisted, see "Attribute: StoreCDRs".

External EDR listeners

External EDR listeners are JMS topic subscribers. See "Creating an EDR Listener and Generating SNMP MIBs" in *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide* for information on how to create a EDR listeners.

EDRService

The following section describes the EDRService.

Configuration of the EDRService

To configure the behavior of the EDRService, in the EdrService managed object:

1. Specify Attribute: PublishToJMS.
2. Specify Attribute: StoreAlarms.
3. Specify Attribute: StoreCDRs.

Management of the 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 Attribute: BatchTimeout.
2. Specify Attribute: BatchSize.

Reference: Attributes and Operations for EDRService

Managed object: Container Services->EdrService

MBean: com.bea.wlcp.wlng.edr.management.EdrServiceMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: BatchTimeout
- Attribute: StatisticsEnabled
- Attribute: BatchSize
- Attribute: PublishToJMS
- Attribute: StoreAlarms
- Attribute: StoreCDRs
- Operation: displayStatistics
- Operation: resetStatistics
- Operation setEdrType

Attribute: BatchTimeout

Scope: Cluster

Format: int

Unit: milliseconds

Specifies the timeout value for a JMS batch.

Attribute: StatisticsEnabled

Scope: Cluster

Format: boolean

Specifies if statistics is enabled for EDRService. Must be enabled for Operation: displayStatistics to be relevant.

Attribute: BatchSize

Scope: Cluster

Format: int

Unit: number of EDRs

Specifies the size of the JMS batch.

Attribute: PublishToJMS

Scope: Cluster

Format: Boolean

Specifies if EDRs shall be published in the JMS topic or not. Needs to be `true` if external EDR listeners are used.

Attribute: StoreAlarms

Scope: Cluster

Format: boolean

Specifies if alarms shall be stored in the database or not.

Attribute: StoreCDRs

Scope: Cluster

Format: boolean

Specifies if CDRs shall be stored in the database or not.

Operation: displayStatistics

Scope: Cluster

Displays a snapshot of the current statistics for EDRService. Attribute: StatisticsEnabled must be `true` for this operation to be relevant.

The following information is displayed:

- Number of EDRs
- Smallest EDR message size in bytes
- Biggest EDR message size in bytes
- Average EDR message size in bytes

Signature:

```
displayStatistics()
```

Operation: resetStatistics

Scope: Cluster

Resets the statistics for the EDRService.

Signature:

```
resetStatistics()
```

Operation setEdrType

Scope: Cluster

Configures the type of EDRs that should be sent from the EDRService.

Signature:

```
setEdrType(Programmable: boolean, Facade: boolean, Enabler, boolean,
ProtocolStack: boolean, Others: boolean)
```

Table 8–2 setEdrType

Parameters	Description
Programmable	true: send EDRs created manually through EdrDataHelper
Facade	true: send EDRs created in the AT layer. Default is false.
Enabler	true: send EDRs created in PluginNorth or PluginSouth via AspectJ
ProtocolStack	true: send EDRs created in PluginNorthCallback, ProtocolServiceNorth, or ProtocolServiceSouth via AspectJ
Others	true: send EDRs created via AspectJ

Managing EDR, CDR, and alarms configuration files using the EDR Configuration Pane

The Oracle Communications Services Gatekeeper EDR Configuration pane allows the administrator to load new EDR, CDR, and alarm configuration files.

Open the pane by selecting **OCSG** > *Server Name* > **EDR Configuration** from the Domain Structure in the Administration Console.

Note: Lock and Edit must be used.

Table 8–3 describes the inputs for the entry fields in the configuration pane.

Table 8–3 Oracle Communications 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".

9

Managing and Configuring Credit Control Interceptors and SLAs

This chapter describes how to manage and configure the Credit Control Interceptors in Oracle Communications Services Gatekeeper.

Introduction

The Credit Control Interceptors performs credit checks towards a Diameter server using the Diameter Ro interface. This allows for easy integration with charging systems that support Diameter Ro on-line charging. Oracle Communications Billing and Revenue Management, can be used out-of-the-box with Oracle Communications Services Gatekeeper.

Traffic flowing through any Communication Service is intercepted and the data in the request, together 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.

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 cancelled.

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 of time, the reservation is cancelled.

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 cancelled.

Only synchronous reservations are supported.

Credit Control Interception Points

Only requests that are explicitly defined to be subject to credit control checks are passed on to the Diameter server.

It is possible to map a parameter in the request to be passed in the Diameter request, or to define a static value for the following AVPs:

- 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 defines which methods that shall be subject to credit control, how the credit control shall be performed, and which data to provide to the Diameter server. Table 9–1 describes the structure and content of a credit control SLA.

Table 9–1 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 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 "Managing and Configuring the Plug-in Manager" 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 9–1 (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. 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. 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. 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 9–1 (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. 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. ■ 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. 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 9–1 (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 "Managing and Configuring the Plug-in Manager" 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. 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 tag <CCInterception>. Normally, the correlator used to correlate asynchronous request-response pairs is used.

Table 9–1 (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. 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 tag <CCInterception>. 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 9–1.

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 "Managing and Configuring the Plug-in Manager" 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.

Example of a static parameter mapping:

```
<OCSGChargeDescription type="static" value ="Static value for reservation."/>
```

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 tag <CCInterception>.

For asynchronous credit control checks, the ID, or correlator, to use to tie together the reservation request and the commit request is defined. 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 9–1 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 9–1 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>
```

Configuration, Management and Provisioning

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.

For more information about service interceptors, see "Service Intercepters" in the *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide*.

The interceptors are configured and managed using an MBean.

The Credit Control SLAs are provisioned using the Service Provider Group and Application Group SLA MBeans, see “Managing SLAs” in the *Oracle Communications Services Gatekeeper Accounts and SLAs Guide*.

Properties for Credit Control Service Interceptors

Table 9–2 describes the properties of Credit Control Service Interceptors.

Table 9–2 Properties

Property	Description
Managed object in Administration Console	<domain name>->OCSG-><server name>->ContainerServices->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 the Credit Control interceptors are not deployed.

Deploy the EAR file that contains the interceptors using WebLogic Server tools. For a description of the different deployment options, see "Deploying, Undeploying, and Redeploying Java EE Applications" in *Oracle® Fusion Middleware Deploying Applications to Oracle WebLogic Server* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13702/title.htm

The EAR file is located in `$OCSG_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.

Configuration of CreditControlInterceptor

To configure the behavior of the CreditControlInterceptor, in the managed object CreditControlInterceptor:

1. Specify:
    - Attribute: PeerRetryDelay
    - Attribute: DestinationHost
    - Attribute: DestinationPort
    - Attribute: DestinationRealm
    - Attribute: OriginHost
    - Attribute: OriginPort
    - Attribute: MoReservationInterceptorIndex
    - Attribute: MtReservationInterceptorIndex
    - Attribute: CommitInterceptorIndex
    - Attribute: PeerRetryDelay
    - Attribute: RequestTimeout
    - Attribute: WatchdogTimeout
2. Use Operation: connect to connect to the Diameter server.

Management of 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:

- Operation: connect
- Operation: disconnect

Use Attribute: Connected (r) to check the current connection status.

Use Operation: connect after any changes to the configuration attributes. Changes do not take affect until this operation is invoked.

Provision Credit Control SLAs

Credit Control SLAs can be enforced on application group and service provider group level.

The SLAs are provisioned as custom SLAs, see “Managing SLAs” in the *Oracle Communications 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 EAR 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`

Reference: Attributes and Operations for CreditControlInterceptor

Managed object: Container Services->CreditControlInterceptor

MBean: com.bea.wlcp.wlng.interceptor.management.DiameterMBean

Following is a list of attributes and operations for configuration and maintenance.

- Attribute: Connected (r)
- Attribute: CommitInterceptorIndex
- Attribute: DestinationHost
- Attribute: DestinationPort
- Attribute: DestinationRealm
- Attribute: OriginHost
- Attribute: OriginPort
- Attribute: PeerRetryDelay
- Attribute: RequestTimeout
- Attribute: WatchdogTimeout
- Operation: connect
- Operation: disconnect

Attribute: Connected (r)

Read-only.

Scope: Server

Format: Boolean

Unit: Not applicable

Displays the status of the connection to the Diameter server.

Displays:

- `true`, if connected.
- `false`, if not connected.

Attribute: CommitInterceptorIndex

Scope: Cluster

Format: Integer

Unit: Not applicable

Specifies where in the Service Interceptor chain to register the Credit Control interceptor for committing asynchronous Credit Control Checks.

Attribute: DestinationHost

Scope: Cluster

Format: String

Unit: Not applicable

Specifies the Destination-Host AVP in the Diameter request.

Can be specified either as an IP-address or a host name.

Attribute: DestinationPort

Scope: Cluster

Format: int

Unit: Not applicable

Specifies the port on the Diameter server to connect to.

Attribute: DestinationRealm

Scope: Cluster

Format: String

Unit: Not applicable

Specifies the Destination-Realm AVP in the Diameter request.

Attribute: OriginHost

Scope: Server

Format: String

Unit: Not applicable

Specifies the Origin-Host AVP in the Diameter request.

Can be specified either as an IP-address or a host name.

Optional.

Attribute: OriginPort

Scope: Server

Format: int

Unit: Not applicable

Specifies the local originator port to be used for the connection to the Diameter server.

If specified as 0 (zero), a random local port is used.

Random port is a requirement in order to support hitless upgrades of the interceptor.

Attribute: MoReservationInterceptorIndex

Scope: Cluster

Format: int

Unit: Not applicable

Specifies where in the Service Interceptor chain to register the Credit Control interceptor for network-triggered requests.

Attribute: MtReservationInterceptorIndex

Scope: Cluster

Format: int

Unit: Not applicable

Specifies where in the Service Interceptor chain to register the Credit Control interceptor for application-initiated requests.

Attribute: PeerRetryDelay

Scope: Cluster

Format: int

Unit: seconds

Specifies the time to wait before attempting to reconnect to the Diameter server if the connection is lost.

Attribute: RequestTimeout

Scope: Cluster

Format: int

Unit: milliseconds

Specifies the maximum time to wait for a response to a request to the Diameter server.

Attribute: WatchdogTimeout

Scope: Cluster

Format: int

Unit: seconds

Specifies the watchdog timeout Tw.

This setting corresponds to the timer Tw, see RFC 3539 at `http://www.ietf.org/rfc/rfc3539.txt`.

It specifies the time to wait before attempting to reconnect to the Diameter server in the case of a lost connection.

Operation: connect

Scope: Cluster

Connects to the Diameter server.

Once connected, the interceptor will try to reconnect to the Diameter server if the server is restarted or the interceptor is redeployed.

Signature:

```
connect()
```

Operation: disconnect

Scope: Cluster

Disconnects from the Diameter server.

Once disconnected, the interceptor will not try to reconnect to the DIAMETER server if the server is restarted or the interceptor is redeployed.

Signature:

```
disconnect()
```

10

Setting Up Geographic Redundancy

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

Introduction

The Geographic Redundancy service replicates data between two geographically distant sites so that applications can switch from one site to another (for example, in case of the catastrophic failure of one) and still have all the configuration data (account information, system Service Level Agreements (SLAs), and budgets) necessary for SLA enforcement available at the second, remote, site. For more information about geographic redundancy, see "Redundancy, Load Balancing, and High Availability" in *Oracle Communications Services Gatekeeper Concepts Guide*.

The sites are set up in pairs. One member of the pair is designated the geomaster, the other the slave. Each geographic site has a name which is used for looking up data relevant to the site pair. The name of the remote site is defined in the local site.

Configuration Workflow

There are two stages to configuring basic geographic redundancy. Both must be done at each site.

- Configure the geographically redundant service.
- Define the geomaster site

Configure Each Site for Geo-Redundancy

To use geographic redundancy, each site must be appropriately configured. This is accomplished using the GeoRedundantService. Using this service, you:

- Define the ID of the local site in Attribute: GeoSiteId.
- Define the number of failed attempts to reach a remote site before an alarm should be raised in Attribute: RemoteSiteReachabilityAlarmThreshold.
- Define the remote site in Operation: setSiteAddress.

Define the GeoMaster Site

One site of the site pair must be designated the geomaster site. This is accomplished using the GeoStorageService. Using this service, you:

- Define the site that is to serve as the geomaster in Attribute: GeoMasterSiteId

Reference: Attributes and Operations for GeoRedundantService

Managed object: Container Services->GeoRedundantService

MBean Type:
com.bea.wlcp.wlng.core.budget.management.configuration.GeoRedundantServiceMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: GeoSiteId
- Attribute: RemoteSiteReachabilityAlarmThreshold
- Operation: getSiteAddress
- Operation: listRemoteSites
- Operation: removeSite
- Operation: setSiteAddress

Attribute: GeoSiteId

Scope: Cluster

Format: String

Defines the name of this geographic redundant site. Must be done at both sites. This name is used as key for all operations on the remote site. See "Operation: setSiteAddress", "Operation: getSiteAddress", "Operation: removeSite".

Attribute: RemoteSiteReachabilityAlarmThreshold

Scope: Cluster

Format: int

Specifies the number of attempts made by a site to reach its peer site before raising an alarm. Must be done at both sites.

Whenever the peer sites fail to establish a connection the number of times defined in `RemoteSiteReachabilityAlarmThreshold`, a connection lost alarm is raised.

Operation: getSiteAddress

Scope: Cluster

Signature:

```
getSiteAddress(Site name: String)
```

Displays the address of a given remote site. Table 10–1 describes this parameter.

Table 10–1 getSiteAddress

Parameter	Description
Site name	The name of the remote site

Operation: listRemoteSites

Scope: Cluster

Signature:

```
listRemoteSites()
```

Displays a list of registered remote sites.

Operation: removeSite

Scope: Cluster

Signature:

```
removeSite(Site name: String)
```

Removes a site definition for a remote site. If both sites are operational, must be done at both sites.

Table 10–2 describes this parameter.

Table 10–2 removeSite

Parameter	Description
Site name	The site name of the remote site

Operation: setSiteAddress

Scope: Cluster

Signature:

```
setSiteAddress(Site name: String, Address: String)
```

Specifies the address of a remote site. Must be done at both sites.

Table 10–3 describes the parameters.

Table 10–3 setSiteAddress

Parameter	Description
Site name	Name of the remote site
Address	JNDI URL of the Network Tier for the remote site, according to WebLogic Server addressing standards: *protocol*://*host*:*port* Example: `t3://host1:port,host2:port`

Reference: Attributes and Operations for GeoStorageService

Managed object: Container Services->GeoStorageService

MBean Type: com.bea.wlcp.wlng.geostorage.management.GeoStorageServiceMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: GeoMasterSiteId
- Operation: syncFromGeoMaster

Attribute: GeoMasterSiteId

Scope: Cluster

Format: String

Defines the geomaster site. This value must be set at both sites and must be one of the two GeoSiteIds set up using the GeoRedundant service. The geomaster keeps the master copy of all geo-configurable data.

Note: If a new site is added to replace a slave site that has failed, it must be added as a slave site. The site that is designated the geomaster site must remain the geomaster site for the lifetime of the site configuration.

If a geomaster site fails permanently, this attribute should be set to empty (temporarily terminating georedundancy) and the failed site should be removed from the configuration using the GeoRedundantService. If a replacement site is added to the configuration, the currently operating site must be the geomaster and the replacement site must be added as the slave.

Operation: syncFromGeoMaster

Scope: Cluster

Signature:

```
syncFromGeoMaster()
```

Forces the slave to resync the account configuration data with the geomaster. Should only be invoked from the slave site. This is used if, for example, the configuration data of the two sites get out of sync, resulting in multiple out-of-sync alarms.

Note: This operation potentially copies large amounts of data and therefore should not be used during peak traffic hours

11

Managing and Configuring Statistics and Transaction Licenses

This chapter describes the statistics functionality and the operation and maintenance procedures for statistics for Oracle Communications Services Gatekeeper.

About Statistics Generation and Reports

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

Statistics are generated only 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 Oracle Communications Services Gatekeeper.

Statistics are held in an in-memory store and are flushed to database at a given time interval. Statistics reports are created based on information in the database.

It is possible to get a snapshot of the current status of the transaction, or request, counters. This information is fetched from the in-memory store.

These report types are available:

- System Report to Console
- System Report to File
- Weekly System Report
- Transaction Usage Log Report

Overview of 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.

System Report to Console

This report is created by Operation: getSystemStatistics. The output is presented in the console.

System Report to File

This report is created by Operation: saveStatisticsToFile. The format is adapted for programmatic processing of the file.

Weekly System Report

The weekly report is a predefined report. It shows the total number of transactions through Oracle Communications 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.

Transaction Usage Log Report

A transaction usage log (called the license_limit log) can be extracted from Oracle Communications 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 11–1 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 11–2 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 *Oracle Communications Services Gatekeeper Licensing Guide*.

Counter Snapshots

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

Table 11–1 describes the different categories into which the snapshot is organized.

Table 11–1 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

Table 11–1 (Cont.) Snapshot Categories

Counter snapshot category	Description
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 11–2.

Table 11–2 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

Managing StatisticService

The following describes managing the StatisticsService.

Configure the StatisticService

Table 11–3 describes how to configure statistics persistence interval.

Table 11–3 Configuring Statistics Persistence Interval

To configure...	Use
Statistics persistence interval	Attribute: StoreInterval

Configure Statistics Types and Transaction Types

Table 11–4 describes the operations to configure statistics types.

Table 11–4 Operations to Configure StatisticsTypes

Statistics types	Use
Add a new	Operation: addStatisticType
Remove an existing	Operation: removeStatisticType
List existing	Operation: removeStatisticType Operation: listStatisticTypes Operation: listStatisticTypeDescriptors

View In-Flight Statistics counters

Table 11–5 shows the attribute used to configure a subset of the statistics counter.

Table 11–5 Attribute to Configure Subset of Statistics Counter

To Configure...	Use
A subset of the statistics counters	Attribute: CounterSnapshot (r)

Generate Statistics Reports

Table 11–6 shows the operations used to generate statistics reports.

Table 11–6 Operations to Create Statistics Reports

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

Add Usage Thresholds

Table 11–7 shows the attributes used to add usage thresholds.

Table 11–7 Attributes to Add Usage Thresholds

To add a...	Use
Limit alarm threshold for Oracle module-based TUPS	Attribute: ModuleBHTUPSThreshold
Limit alarm threshold for platform-based TUPS	Attribute: PlatformBHTUPSThreshold

Reference: attributes and operations for StatisticsService

Managed object: Container Services->StatisticsService

MBean: com.bea.wlcp.wlng.statistics.management.StatisticsServiceMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: CounterSnapshot (r)
- Attribute: StoreInterval
- Attribute: ModuleBHTUPSThreshold
- Attribute: PlatformBHTUPSThreshold
- Operation: addStatisticType
- Operation: createLicenseLimitLog
- Operation: createWeeklyReport
- Operation: getStatistics

- Operation: getSystemStatistics
- Operation: listStatisticTypeDescriptors
- Operation: listStatisticTypes
- Operation: removeStatisticType
- Operation: saveAccountStatisticsToFile
- Operation: saveStatisticsToFile

Attribute: CounterSnapshot (r)

Read only.

Scope: Server

Displays a snapshot with information about all statistics-related counters for the chosen Oracle Communications Services Gatekeeper server.

The information is useful for analyzing runtime traffic and, by invoking it periodically, can be used to get an approximate value of the traffic throughput. See "Counter Snapshots".

> **Note:** The counters are reset periodically.

Attribute: StoreInterval

Unit: seconds.

Scope: Cluster

Specifies how often statistics data is persisted to the database.

Attribute: ModuleBHTUPSThreshold

Unit: int

Scope: Cluster

Specifies the TUPS limit alarm threshold for Oracle modules. The value of 0 means no alarms.

Attribute: PlatformBHTUPSThreshold

Unit: int

Scope: Cluster

Specifies the TUPS limit alarm threshold for platform modules The value of 0 means no alarms.

Operation: addStatisticType

Scope: Cluster

Adds a statistic type. Used to add a statistics type for custom communication services.

Signature:

```
addStatisticType(Id: int, Name: String)
```

Table 11–8 describes these parameters.

Table 11–8 addStatisticType

Parameter	Description
id	Statistic type ID
Name	Statistic type name. Descriptive name for the statistic type.

Operation: createLicenseLimitLog

Scope: Cluster

Creates a transaction usage supervision log that contains information about the busy hour transaction rate for each transaction group and day during a given time period.

Signature:

```
createLicenseLimitLog(filename: String, startDate: String, endDate: String)
```

Table 11–9 describes these parameters.

Table 11–9 createLicenseLimitLog

Parameter	Description
filename	The filename for the report. The file is created on the local file system of the selected server. Must include an absolute path. **Note:** The file must not already exist. If it already exists, the operation will fail.
startDate	Specifies the start date for the report. Format is YYYY-MM-DD.
endDate	Specifies the end date for the report. Format is YYYY-MM-DD.

Operation: createWeeklyReport

Scope: Cluster

Creates a weekly report. See "Weekly System Report" for a description of the report.

Signature:

```
createWeeklyReport(SpAccountId: String, StartDate: String, FileName: String,
Decimals: int)
```

Table 11–10 describes these parameters.

Table 11–10 createWeeklyReport

Parameter	Description
SpAccountId	Service provider ID to filter on. **Note:** Leave empty to include all service provider account in the report.
StartDate	Specifies the start date for the report. Format is YYYY-MM-DD.
FileName	The filename for the report. Must include an absolute path. The file is created on the local file system of the selected server. **Note:** The file must not already exist. If it already exists, the operation will fail.

Table 11–10 (Cont.) createWeeklyReport

Parameter	Description
Decimals	Number of decimals in the entry for transactions/second in the report.

Operation: getStatistics

Scope: Cluster or Server

Displays a detailed statistics report for a specific service provider and application. Wildcards and filters apply.

Signature:

```
getStatistics(ServerName: String, StatisticType: String, fromDate: String, toDate: 
String)
```

Table 11–11 describes these parameters.

Table 11–11 getStatistics

Parameter	Description
ServerName	Specifies the name of the server to display statistics from. Leave empty to display statistics generated in all servers.
StatisticType	Specifies the statistics type ID for the statistics type to display. See "Operation: listStatisticTypes". **Note:** Use -1 to display all statistics types.
fromDate	Specifies the start date and time for the interval to display. Format is YYYY-MM-DD hh:mm. **Note:** Leave empty to display all statistics up to the date and time specified in toDate.
toDate	Specifies the end date and time for the interval to display. Format is YYYY-MM-DD hh:mm. **Note:** Leave empty to display all statistics generated from the date and time specified in fromDate up to current date and time.
spAccountId	Service provider account ID to filter on. Leave empty to wildcard.
appAccountId	Application account ID to filter on. Leave empty to wildcard.

Operation: getSystemStatistics

Scope: Cluster or Server

Displays a summary of system statistics for the last minutes. See "Overview of Statistics Reports" for information on output format.

Signature:

```
getSystemStatistics(minutes: int)
```

Table 11–12 describes this parameter.

Table 11–12 getSystemStatistics

Parameter	Description
minutes	The number of minutes relative to the current time.

Operation: listStatisticTypeDescriptors

Scope: Cluster

Displays a list of all available statistics type descriptors. The descriptors contain information on transactionTypeName and transactionTypeID.

Signature:

```
listStatisticTypeDescriptors()
```

Operation: listStatisticTypes

Scope: Cluster

Specifies the statistics types included in the statistics reports.

Signature:

```
listStatisticTypes()
```

Operation: removeStatisticType

Scope: Cluster

Removes a statistics type.

Signature:

```
removeStatisticType(statisticsType: int)
```

Table 11–13 describes this parameter.

Table 11–13 removeStatistics

Parameter	Description
statisticsType	ID for the statistics type.

Operation: saveAccountStatisticsToFile

Scope: Cluster/Server

Saves a statistics report to a file. Filters can be applied on service provider ID and application ID. The report is grouped by account.

Signature:

```
saveAccountStatisticsToFile(StatisticType: String, fromDate: String, toDate:
String, serviceProviderIdentifier String, applicationIdentifier: String, filename:
String)
```

Table 11–14 describes these parameters.

Table 11–14 saveAccountStatisticsToFile

Parameter	Description
StatisticType	ID for the statistics type. **Note:** Use -1 to display all statistics types.
fromDate	Specifies the start date and time for the time interval. Format is YYYY-MM-DD hh:mm. **Note:** Leave empty for all statistics up to the date and time specified in `toDate`.

Table 11–14 (Cont.) saveAccountStatisticsToFile

Parameter	Description
toDate	Specifies the end date and time for the time interval. Format is YYYY-MM-DD hh:mm. **Note:** Leave empty for all statistics generated from the date and time specified in `fromDate` up to current date and time.
serviceProviderIdentifier	ID of the service provider to filter on. Leave empty for no filtering.
applicationIdentifier	ID of the application to filter on. Leave empty for no filtering.
filename	Specifies the filename for the report. Must include an absolute path. The file is created on the local file system of the selected server. **Note:** The file must not already exist. If it does already exist, the method will fail.

Operation: saveStatisticsToFile

Scope: Cluster/Server

Saves a statistics report to file.

Signature:

```
saveStatisticsToFile(serverName: String, StatisticType: String, fromDate: String,
toDate: String, filename: String)
```

Table 11–15 describes these parameters.

Table 11–15 saveStatisticsToFile

Parameter	Description
serverName	Specifies the name of the server. Leave empty for statistics generated in all servers.
StatisticsType	ID for the statistics type. **Note:** Use -1 to display all statistics types.
fromDate	Specifies the start date and time for the time interval. Format is YYYY-MM-DD hh:mm. **Note:** Leave empty for all statistics up to the date and time specified in `toDate`.
toDate	Specifies the end date and time for the time interval. Format is YYYY-MM-DD hh:mm. **Note:** Leave empty for all statistics generated from the date and time specified in `fromDate` up to current date and time.
FileName	Specifies the filename for the report. Must include an absolute path. The file is created on the local file system of the selected server. **Note:** The file must not already exist. If it does already exist, the method will fail.

Transaction Types

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

Which events generate which statistics for a certain transaction type are described in *Oracle Communications Services Gatekeeper Communication Service Guide,* in the discussion of statistics for the communication service in question.

The information includes correlation maps between methods being invoked from either an application or the telecom network and the corresponding transaction type. See *Oracle Communications Services Gatekeeper Communication Service Guide*.

12

CDRs and Diameter

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

About CDRs and Diameter

The CdrToDiameter service forwards charging events to a Diameter server using the DIAMTER Rf interface. This allows for easy integration with charging systems that support Diameter Rf offline charging. When enabling this service, Oracle Communications Billing and Revenue Management, together with Oracle Communications Network Mediation, can be used out-of-the-box with Oracle Communications Services Gatekeeper.

Any traffic request processed through Oracle Communications 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 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 "Deploying Shared Java EE Libraries and Dependent Applications" in Java EE 5 Deployment Implementation at *Oracle® Fusion Middleware Deploying Applications to Oracle WebLogic Server* at:

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13702/understanding.htm`.

The service is packaged in `cdr_to_diameter-single.ear` and `cdr_to_diameter.ear` in `$OCSG_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, and is transferred to another server in case of server failure.

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:
   - Attribute: OriginHost
   - Attribute: OriginPort
   - Attribute: DestinationHost
   - Attribute: DestinationPort
   - Attribute: DestinationRealm
   - Attribute: PeerRetryDelay
   - Attribute: RequestTimeout
   - Attribute: WatchdogTimeout
2. Use Operation: connect 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:

- Operation: connect
- Operation: disconnect

Use Attribute: Enabled (r) to check the current connection status.

Use Operation: connect after any changes to the configuration attributes. Changes does not take affect until this operation is invoked.

Reference: Attributes and Operations for CdrToDiameter

Managed object: Container Services->CdrToDiameter

MBean: com.bea.wlcp.wlng.cdrdiameter.management.CDRDiameterMBean

Following is a list of attributes and operations for configuration and maintenance.

- Attribute: Enabled (r)
- Attribute: OriginHost
- Attribute: OriginPort
- Attribute: DestinationHost
- Attribute: DestinationPort
- Attribute: DestinationRealm
- Attribute: PeerRetryDelay
- Attribute: RequestTimeout
- Attribute: WatchdogTimeout
- Operation: connect

- Operation: disconnect

Attribute: Enabled (r)

Read-only.

Scope: Cluster

Format: Boolean

Unit: Not applicable

Displays the status of the CdrToDiameter service.

Displays:

- `true`, if enabled.
- `false`, if not enabled.

Attribute: OriginHost

Scope: Server

Format: String

Unit: Not applicable

Specifies the Origin-Host AVP in the Diameter request.

Can be specified either as an IP-address or a host name.

Optional.

Attribute: OriginPort

Scope: Server

Format: int

Unit: Not applicable

Specifies the local originator port to be used for the connection to the Diameter server.

If specified as 0 (zero), a random local port is used.

Attribute: DestinationHost

Scope: Server

Format: String

Unit: Not applicable

Specifies the Destination-Host AVP in the Diameter request.

Can be specified either as an IP-address or a host name.

Attribute: DestinationPort

Format: int

Unit: Not applicable

Specifies the port on the Diameter server to connect to.

Attribute: DestinationRealm

Scope: Cluster

Format: String

Unit: Not applicable

Specifies the Destination-Realm AVP in the Diameter request.

Attribute: PeerRetryDelay

Scope: Cluster

Format: int

Unit: seconds

Specifies the time to wait before attempting to reconnect to the Diameter server if the connection is lost.

Attribute: RequestTimeout

Scope: Cluster

Format: int

Unit: milliseconds

Specifies the maximum time to wait for a response to a request to the Diameter server.

Attribute: WatchdogTimeout

Scope: Cluster

Format: int

Unit: seconds

Specifies the watchdog timeout Tw.

This setting corresponds to the timer Tw, see RFC 3539 at

http://www.ietf.org/rfc/rfc3539.txt

It specifies the time to wait before attempting to reconnect to the Diameter server in the case of a lost connection.

Operation: connect

Scope: Cluster

Connects to the Diameter server.

Once connected, the service will try to reconnect to the Diameter server if the server is restarted or the service is redeployed.

Signature:

```
connect()
```

This operation accepts no parameters.

Operation: disconnect

Scope: Cluster

Disconnects from the Diameter server.

Once disconnected, the service will not try to reconnect to the DIAMETER server if the server is restarted or the service is redeployed.

Signature:

```
disconnect()
```

This operation accepts no parameters.

CDR to AVP mapping

The CDRs that are generated are mapped to Diameter attribute-value pairs, AVPs, as shown in Table 12–1. Both standard and custom AVPs are used. When custom are used it is indicated in the table.

Table 12–1 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
Origin-Host	Configurable, see "Attribute: OriginHost".	Realm of the Oracle Communications 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 Oracle Communications Services Gatekeeper domain. Addressed with the domain address of the corresponding public URI. Specification: RFC 3588 Example: oracle.com
Destination-Realm	Configurable, see "Attribute: DestinationRealm".	Realm of the operator domain. Addressed with the domain address of the corresponding public URI. Specification: RFC 3588 Example: oracle.com

Table 12–1 (Cont.) CDR to Diameter AVP Mappings

AVP	Source	Description
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
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

Table 12–1 (Cont.) CDR to Diameter AVP Mappings

AVP	Source	Description
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.
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

Table 12–1 (Cont.) CDR to Diameter AVP Mappings

AVP	Source	Description
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

13

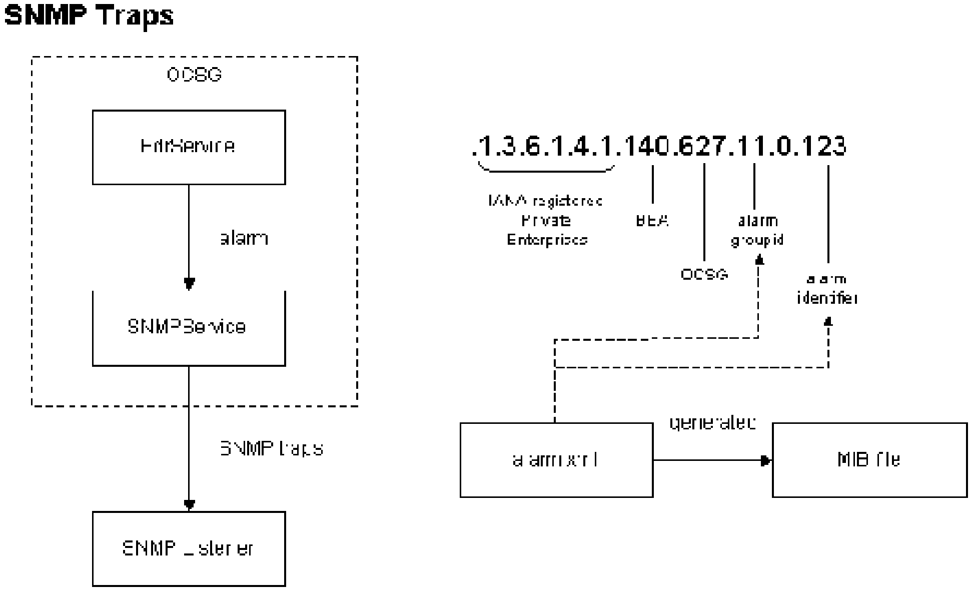
Managing and Configuring the SNMP Service

This chapter describes how to configure and manage the Simple Network Management Protocol (SNMP) service.

Introduction

The SNMP service is responsible for collecting alarms and distributing them as SNMP traps.

Figure 13–1 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 1:1 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.

Each individual alarm ID is used to generate the object identifier for each SNMP trap.

The SNMP traps sent consist of:

IANA-registered private enterprise ID + BEA ID + Oracle Communications Services Gatekeeper ID + “0” + alarm identifier

- The IANA-registered enterprise ID is configurable using Attribute: EnterpriseObjectIdentifier.
- The BEA ID is static, and the value is 140.
- The Oracle Communications 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 `BEA-WLNG-MIB` MIB file is located in the *$OCSG_Home*/**ocsg_pds_5.1/integration** directory.

Configuration and management

The following section describes the SNMPService.

Configure SNMPService

Table 13–1 lists the attributes used to configure the SNMP Service.

Table 13–1 Task Overview

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

Define trap receivers using Operation: addTrapReceiver.

Trap Receivers

Table 13–2 lists the operations on trap receivers.

Table 13–2 Task overview

To	Use
Add a trap receiver	Operation: addTrapReceiver
List defined trap receivers	Operation: listTrapReceivers
Delete an existing trap receiver	Operation: listTrapReceivers to list registered trap receivers. Use the ID as an input parameter in Operation: deleteTrapReceiver.

Reference: Attributes and Operations for SNMPService

Managed object: Container Services->SNMPService

MBean: com.bea.wlcp.wlng.snmp.SNMPServiceMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: Community

- Attribute: EnterpriseObjectIdentifier
- Attribute: RepeatedTraps
- Attribute: SNMPVersion
- Attribute: SeverityFilter
- Operation: addTrapReceiver
- Operation: deleteTrapReceiver
- Operation: listTrapReceivers

Attribute: Community

Scope: Server

Format: String

Units: Not applicable

Specifies the SNMP community address.

Default value is private.

Attribute: EnterpriseObjectIdentifier

Scope: Server

Format: String

Units: Not applicable

Specifies the base enterprise object identifier used for the SNMP traps.

BEA ID and Oracle Communications Services Gatekeeper IDs are fixed and appended to this ID.

For each individual SNMP trap, the alarm ID for the alarm is appended per alarm.

Attribute: RepeatedTraps

Scope: Cluster

Format: int

Units: Not applicable

Specifies the number of times each SNMP trap is sent to each configured manager.

Attribute: SNMPVersion

Scope: Cluster

Format: int

Units: Not applicable

Specifies the SNMP version to use.

Enter:

- 0 to use SNMP v1.
- 1 to use SNMP v2.

Attribute: SeverityFilter

Scope: Cluster

Format: int

Units: Not applicable

Specifies the severity filter setting.

Only alarms with a severity that exceeds or is equal to the specified severity filter will cause a trap to be generated.

Enter:

- 4 to generate SNMP traps for CRITICAL alarms.
- 3 to generate SNMP traps for MAJOR and more severe alarms.
- 2 to generate SNMP traps for MINOR and more severe alarms.
- 1 to generate SNMP traps for WARNING and more severe alarms.

Operation: addTrapReceiver

Scope: Cluster

Adds a receiver for the SNMP traps. Returns an ID for the receiver.

Signature:

```
addTrapReceiver(Address: String, Port: int)
```

Table 13–3 describes these parameters.

Table 13–3 addTrapReceiver

Parameter	Description
Address	IP address for the trap receiver
Port	Port for the trap receiver

Operation: deleteTrapReceiver

Scope: Cluster

Deletes a previously added trap receiver.

Signature:

```
deleteTrapReceiver(id: int)
```

Table 13–4 describes this parameter.

Table 13–4 deleteTrapReceiver

Parameter	Description
id	ID of the trap receiver to delete. Use Operation: listTrapReceivers to list the IDs.

Operation: listTrapReceivers

Scope: Cluster

Displays a list of all registered trap receivers.

Signature:

```
listTrapReceivers()
```

This operation accepts no parameters.

14

Managing and Configuring the Trace Service

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

Introduction to the Trace Service

The Trace service is based on Log4J. Oracle Communications Services Gatekeeper maintains the log file, named **default.log**. Each service instance writes to this log file, the local file system of the server on which it executes. The trace service writes log files in the *Domain_home*/**servers/***server name*/**trace** directory.

Basic tracing

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

Following is a list of attributes and operations for configuration and maintenance for basic tracing:

- Attribute: TracingEnabled
- Operation: attachAppender
- Operation: flushBuffers
- Operation: rollOver

Context trace

Context tracing generates log messages filtered on the context of a request, in addition to basic tracing; for example, for a certain service provider or application. 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 Oracle Communications 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 are added to the context trace file. 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.

The workflow for defining a context trace file is:

1. In the Oracle Communications Services Gatekeeper Administration Console, choose **Container Services** >**TraceService**.

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

Reference: Attributes and Operations for Trace Service

Managed object: Container Services->TraceService

MBean: com.bea.wlcp.wlng.log.management.TraceServiceMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: TracingEnabled
- Operation: addContextCategory
- Operation: addContextFilter
- Operation: attachAppender
- Operation: createContextTraceFile
- Operation: createRootContextTraceFile
- Operation: removeContextTraceFile
- Operation: resetContextFilters
- Operation: rollOver

Attribute: TracingEnabled

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether trace should be enabled or not.

Use:

- `true` to enable tracing
- `false` to disable tracing

Operation: addContextCategory

Scope: Cluster

Adds a Log4J context category.

Signature:

```
addContextCategory(Identifier: String, Category: String)
```

Table 14–1 describes these parameters.

Table 14–1 addContextCategory

Parameter	Description
Identifier	ID of a context trace file.

Table 14–1 (Cont.) addContextCategory

Parameter	Description
Category	The Log4J category. The category is the package name, including subpackages, to trace on. For example, to trace on the SMPP plug-in, enter `com.bea.wlcp.wlng.plugin.sms.smpp`.

Operation: addContextFilter

Scope: Cluster

Adds one of the predefined filter types to the appender with the identified name.

The filter is a name/value pair, where the type identifies the filter type to use, and the value is the value of the filter.

All trace information that matches the filter is written to the context trace file.

When the Trace service receives information about a request, it checks if the request matches the context filter. If it matches, the trace information is written to file. All filters must match for the trace to be written.

Examples:

To add a filter that matches all requests from service provider ID SP1, set the type to *SERVICE_PROVIDER* and the value to *SP1*.

To add a filter that matches all requests from application ID APP1, set the type to *APPLICATION* and the value to *APP1*.

Signature:

```
addContextFilter(identifier: String, type: String, value: String)
```

Table 14–2 describes these parameters.

Table 14–2 addContextFilter

Parameter	Description
Identifier	ID of a context trace file.
Type	One of the defined types to filter on. Use: ■ SERVICE_PROVIDER to filter on service provider ID. ■ APPLICATION to filter on application ID. ■ APPLICATION_INSTANCE to filter on application instance ID. ■ SESSION to filter on session ID. ■ TRANSACTION to filter on transaction ID.
Value	Value to filter on for this filter type.

Operation: attachAppender

Scope: Cluster

Allows adding a named appender to named loggers. Each Oracle Communications Services Gatekeeper internal service has a default FileAppender that outputs to a file with the same name as the Oracle Communications Services Gatekeeper internal service.

Signature:

```
attachAppender(LoggerName: String, AppenderName: String)
```

Table 14–3 describes these parameters.

Table 14–3 attachAppender

Parameter	Description
LoggerName	Name of logger.
AppenderName	Name of Oracle Communications Services Gatekeeper internal service to attach the appender to.

Operation: createContextTraceFile

Scope: Cluster

Creates a new context trace file.

Signature:

```
createContextTraceFile(Identifier : String, Category: String, Threshold: String)
```

Table 14–4 describes these parameters.

Table 14–4 createContextTraceFile

Parameter	Description
Identifier	ID of the context trace file to create. Also the name of the trace file. The filename has the suffix `.log`. Must be unique per managed server.
Category	The Log4J category.
Threshold	The threshold for the context trace file. Valid values are: ▪ OFF ▪ FATAL ▪ ERROR ▪ WARN ▪ INFO ▪ DEBUG ▪ TRACE ▪ ALL

Operation: createRootContextTraceFile

Scope: Cluster

Creates a new root context trace file. Adds a context trace file under root trace directory and associates it with the root logger

Signature:

```
createRootContextTraceFile(Identifier: String)
```

Table 14–5 describes this parameter.

Table 14–5 createRootContextTraceFile

Parameter	Description
Identifier	ID of the root context trace file. Also the name of the trace file. The filename has the suffix `.log`. Must be unique per managed server.

Operation: flushBuffers

Scope: Cluster

Flushes trace buffers to file. This operation affect only FileAppenders or subclasses.

Signature:

```
flushBuffers()
```

Operation: removeContextTraceFile

Scope: Cluster

Removes a context trace file.

Signature:

```
removeContextTraceFile(Identifier: String)
```

Table 14–6 describes this parameter.

Table 14–6 removeContextTraceFile

Parameter	Description
Identifier	ID of the context trace file to remove.

Operation: resetContextFilters

Scope: Cluster

Resets all filters associated with a context trace file.

Signature:

```
resetContextFilters(Identifier: String)
```

Table 14–7 describes this parameter.

Table 14–7 resetContextFilters

Parameter	Description
Identifier	ID of the context trace file to reset all filters for.

Operation: rollOver

Scope: Cluster

Rotates log files. This will invoke the `rollOver()` method on all registered RollingFileAppenders.

Signature:

```
rollOver()
```

Log4J Hierarchies, Loggers, and Appenders

There is a set of Log4J Dynamic MBeans shipped with Oracle Communications Services Gatekeeper that come by default with appenders. One Log4J hierarchy defined and one location are defined and displayed as follows in the following entries in the Oracle Communications Services Gatekeeper Administration Console under **Log4J**:

- log4j:hiearchy=default,Location=AdminServer

The rootDefault logger is connected, to the default hierarchy:

- log4j:Location=AdminServer,logger=rootDefault

Additional loggers can be added, and additional appenders can be added to the loggers. It is possible to change both the priority of the logger and the appender to use. Parameters for the loggers and appenders can be configured.

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

Unlike trace configuration for Network Tier servers, which is performed through the Oracle Communications Services Gatekeeper Administration Console, in *Server Name* > **Log4J**, 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 Oracle Communications 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 by default an empty skeleton.

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:

Everything under:

- `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 14–1 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>
```

15

Managing and Configuring the Storage Service

This chapter describes how to configure and manage the Storage service.

Introduction to the Storage Service

The Storage service provides transparent data access to communication services and container services. Conceptually, a module uses the Storage service, which 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 15–1 outlines the store types.

Table 15–1 Store types

Type	Description
Cluster cache	A cluster-wide 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 cluster-wide 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 cluster-wide 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 will be slow compared to the cluster store, but read operations will be fast if the data is available in cache. This store offers best reliability.

Table 15–1 (Cont.) Store types

Type	Description
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, but 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**

The store definitions are JAR files with a configuration file and class definitions for the stored data.

Storage Data Expiration

To prevent the databases underlying the Storage service from growing too large, data in the Storage service can be set to expire, after which time the data is deleted. One of the visible consequences of this is that the status of delivery receipt can be unsuccessful if an old message is deleted from storage because it aged beyond the configured expiration period.

You can configure both the data expiration period for a communication service and the interval that Services Gatekeeper checks whether data in the stores has expired.

Configuring the Expiration Period

To set the data expiration period:

1. Locate the store schema JAR file for the store that you want to configure.

   In Services Gatekeeper 5.1, 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>
```

Configuring the Checking 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 every 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"
```

Reference: Attributes and Operations for Storage Service

Managed object: Container Services->StorageService

MBean: com.bea.wlcp.wlng.storage.management.StorageServiceMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: Active (r)
- Operation: getProviderName
- Operation: getStoreSize
- Operation: getTableName
- Operation: getTypeId
- Operation: listStoreNames

Attribute: Active (r)

Scope: Cluster

Read-only.

Unit: Not applicable

Format: Boolean

Indicates whether the storage service is active:

- `true` if active
- `false` if inactive

Operation: getProviderName

Scope: Cluster

Gets the storage provider name for a given store.

Signature:

```
getProviderName(StoreName: String)
```

Table 15–2 describes this parameter.

Table 15–2 getProviderName

Parameter	Description
StoreName	The name of the store as specified in the store configuration file.

Operation: getStoreSize

Scope: Cluster

Gets the size of a given store. The size is number of entries.

Signature:

```
getStoreSize(StoreName: String)
```

Table 15–3 describes this parameter.

Table 15–3 getStoreSize

Parameter	Description
StoreName	The name of the store as specified in the store configuration file.

Operation: getTableName

Scope: Cluster

Gets the name of the database table that the store maps to, if any.

Signature:

```
getTableName(StoreName: String)
```

Table 15–4 describes this parameter.

Table 15–4 getTableName

Parameter	Description
StoreName	The name of the store as specified in the store configuration file.

Operation: getTypeId

Scope: Cluster

Gets the type ID of the store.

Signature:

```
getTypeId(StoreName: String)
```

Table 15–5 describes this parameter.

Table 15–5 getTypeId

Parameter	Description
StoreName	The name of the store as specified in the store configuration file.

Operation: listStoreNames

Scope: Cluster

Displays a list of configured store names.

Signature:

```
listStoreNames()
```

16

Securing Web Services and OAM MBeans

This chapter describes enabling authentication settings for REST- and SOAP-based Web services and for securing the Oracle Access Manager (OAM) MBeans in Oracle Communications Services Gatekeeper (Services Gatekeeper).

Introduction

One of the first things you must do in 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 to create a secure Services Gatekeeper implementation, see *Oracle Communications 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, Oracle Communications Services Gatekeeper is preconfigured to send credentials in clear text and this is enforced only for inbound traffic. Outbound security is not enabled because there is no way for Services Gatekeeper to know what security policies may be required by a client. Of course, as soon as you find out what your clients require you should set up that level of security.

Oracle recommends that for production implementations you fully secure Web services using the instruction in one of these sections of the Oracle Communications Services Gatekeeper System Administrator's Guide, 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"

You probably want to 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. If you wish to secure the link by which Oracle Communications Services Gatekeeper returns notifications, you should 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, the mechanism that allows OAM procedures to be performed.

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

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 X509 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 need to configure other web services to use this strategy. See "Setting up UsernameToken with Password Digest" of the Oracle Communications Services Gatekeeper System Administrator's Guide 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 *Oracle Fusion Middleware Security and Administrator's Guide for Web Services* for details at:

  http://download.oracle.com/docs/cd/E15523_01/web.1111/b32511/weblogic.htm

- 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), so if the message is altered en route, the signature becomes invalid. See "Setting up UsernameToken with X.509" of the Oracle Communications Services Gatekeeper System Administrator's Guide for instructions.

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

- "Securing and Administering WebLogic Web Services" in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services* at:

  http://download.oracle.com/docs/cd/E15523_01/web.1111/b32511/weblogic.htm

- "Understanding Oracle WSM Policy Framework" in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services* at: http://download.oracle.com/docs/cd/E15523_01/web.1111/b32511/toc.htm
- *Web Services Security specifications* http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

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 Digest to a Web Service endpoint in Oracle Communications 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** checkbox.
   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 16–1 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 16–1 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 Fusion Middleware Securing Oracle WebLogic Server* at

   `http://download.oracle.com/docs/cd/E15523_01/web.1111/e13707/toc.htm`

   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. Oracle Communications 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 Oracle Communications Services Gatekeeper for the first time. Certain configuration values are cached on start up. So, for example, if you started Oracle Communications Services Gatekeeper during the post-install phase to set up additional JMS Servers, you need to 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. In the following example, this would involve deleting the <port-policy> element.

> **Note:** See Example 16–2 and Example 16–3 for before and after snippets.

Example 16–2 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 16–3 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>
```

1. Re-package the war file with the modified `weblogic-webservices-policy.xml` file
2. Re-package the ear file.
3. Copy the modified ear file to the domain directory of the Administration Server.
4. If you have previously run Oracle Communications Services Gatekeeper, you should now re-deploy the EAR file using the Administration Console.

Create and use a custom WS-Policy

For information about creating and using a custom policy file for message-level security, the discussion on securing Web services in *Oracle® Fusion Middleware Security and Administrator's Guide for Web Services* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/b32511/toc.htm

There is also information about this in the *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help* at:

http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e13952/core/index.html

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, Oracle Communications 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.

Securing RESTful Web Services with SSL

Services Gatekeeper uses SSL authentication to secure REST-based Web services. You have the options listed in Table 16–1 for security levels.

Table 16–1 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.

Table 16–1 (Cont.) SSL Security Options

SSL Level	Client Response Setting	Notes
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 16–2 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 16–2 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**
Default trust keystore location	*Oracle_home*/**Middleware/wlserver_10.3/server/lib/DemoTrust.jks**
Default trust keystore password	**DemoTrustKeyStorePassPhrase**

The following section explain how to configure SSL security for the REST-based Web services.

Configuring Application-Facing Servers for SSL

Follow the instructions in the following sections of the Oracle Communications Services Gatekeeper System Administrator's Guide to set up and configure SSL security for RESTful communication between Services Gatekeeper and external applications:

- Enable and Configure SSL for Each Application Tier Server
- Add Certificates to the Application Tier Servers and Applications

Enable and Configure 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** tap, then the **General** subtab.

4. Check the **SSL Listen Port Enabled** checkbox.
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. You can use the default values (listed in Table 16–2 for test and evaluation implementations:
   1. **Keystores**; Click Change.
   2. **Demo Identity Keystore:** - Change this to different location that you have created.
   3. **Demo Identity Keystore Passphrase:** - Change this to a secure password that you create.
   4. **Demo Trust Keystore:** - Change this to a location that you have created.
   5. **Demo Trust KeyStore Pass phrase:** - Change this to a secure password the you create.
8. Navigate to the SSL
9. If you are setting up a production environment, change these settings to values that you create. You can use the default values (listed in Table 16–2 for test and evaluation implementations:
   1. **Identity and Trust Location:**; Click **Change**.
   2. **Private key Passphrase** - Change this to a secure password that you choose.
   3. **Two Way Client Cert Behavior:** - See Table 16–1 for the possible values. Select the value appropriate for your implementation.
10. Click **Release Configuration.**

Add Certificates to the Application Tier Servers and Applications

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

1. Open a shell on the Services Gatekeeper server.
2. Change directory to the location of your trust store. You set this with the directions in the "Enable and Configure SSL for Each Application Tier Server" section of the Oracle Communications Services Gatekeeper System Administrator's Guide. The default directory is *Oracle_home*/**Middleware/wlserver_10.3/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 "Enable and Configure SSL for Each Application Tier Server" of the Oracle Communications Services Gatekeeper System Administrator's Guide.

keystore_password is the keystore password you created in "Enable and Configure SSL for Each Application Tier Server" of the Oracle Communications Services Gatekeeper System Administrator's Guide.

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. 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 directory to the location of your keystore. You set this with in the Oracle Communications Services Gatekeeper System Administrator's Guide section "Enable and Configure SSL for Each Application Tier Server" tasks. The default directory is *Oracle_home*/**Middleware/wlserver_10.3/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 "Enable and Configure SSL for Each Application Tier Server" of the Oracle Communications Services Gatekeeper System Administrator's Guide.

truststore_password is the truststore password you created in "Enable and Configure SSL for Each Application Tier Server" of the Oracle Communications Services Gatekeeper System Administrator's Guide.

Securing Network-Facing Servers With Keystores

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

- 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 "Add Certificates to the Application Tier Servers and Applications" of the Oracle Communications Services Gatekeeper System Administrator's Guide.

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 Oracle Communications Services Gatekeeper System Administrator's Guide

section "Enable and Configure SSL for Each Application Tier Server" tasks. The default directory is *Oracle_home*/**Middleware/wlserver_10.3/server/lib**.

3. Import the network server keystore credentials, generate a public/private key pair, and create a keystore file with 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. Import the Services Gatekeeper keystore credentials into the network node server with 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** checkbox.

8. Navigate to **wlng**, then **RESTfulClientSSL**.

9. Edit these fields to direct Services Gatekeeper to use the keystores you have imported:

   - **KeyStoreFile:** - The location of the *keystore_file*.**jks** file you imported in step 3.
   - **KeyStorePassPhrase:** - The *publk_key_password* you used in step 3.
   - **KeyStoreType:** - Enter **JKS**.
   - **PrivateKeyPhrase:** - 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 Oracle Communications Services Gatekeeper System Administrator's Guide section "Managing Management Users and Management User Groups". 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 create JMX policies in *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help* at:

`http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e13952/core/index.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 wish 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. Oracle Communications 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 Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13747/xacmlref.htm

For a reference for XACML on WebLogic Server, see “A Reference for XACML on WebLogic Server” in *Oracle Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13747/xacmlref.htm

17

Deployment Model for Communication Services and Container Services

This chapter describes the deployment model used in Oracle Communications Services Gatekeeper. Communication services are packaged and deployed in EAR files. All network protocol plug-ins that share the same group of application-facing interfaces are packaged into the same EAR file.

Container services are packaged and deployed separately from the communication services and should not be modified.

Packaging of Communication Services

All communication services that share a common set of application-facing interfaces are grouped together. For example, Oracle Communications 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,** which serves as the *service facade*. This part consists only of modules shared among the communication service. The **wlng_at_***communication service***.ear** is deployed in the Access Tier.

wlng_nt_*communication service***.ear**, which serves as the *service enabler*. This part contains the network protocol plug-ins for the communication service and common modules for the communication service. The **wlng_nt_***communication service***.ear** is deployed in the Network Tier.

The communication services EAR files are located in the *Middleware_Home***/ocsg_5.1/applications** directory.

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**

wlng_nt_third_party_call_px21.ear 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. See note below.

Other modules in this 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.

> **Note:** **Plugin_px21_third_party_call_sip.jar** is not the only artifact 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. In addition, for plug-ins that use HTPP as the transport protocol and handle network-triggered messages from the network node, there is an HTTP servlet. HTTP servlets are packaged as Web Archives (WARs) and are packaged in the network tier EAR for the communication service. 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**.

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 17–1describes the elements in **wlng_nt_***communication service***.ear**

Example 17–1 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
```

All plug-ins in the service enabler are packaged as individual jar files in the root of the EAR together 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.

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 and 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.

For more information about enterprise application deployment descriptor elements, see "Enterprise Application Deployment Descriptor Elements" in *Oracle® Fusion Middleware Developing Applications for Oracle WebLogic Server* at

`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13706/app_xml.htm`

Example 17–2 describes the elements in a plug-in jar file.

Example 17–2 Contents of <plug-in X>.jar

```
+---com/
+---org/
+---META-INF/MANIFEST.MF
+   instancemap
+   srv_depl.xml
```

The jar contains the regular elements in a jar and also an `instancemap`, used among other thing to instantiate the plug-in specific implementation of the plug-in interface using the InstanceFactory. See "Class: InstanceFactory" in the *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide* for more information.

Deployment of 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 uses a single cluster of AT servers, this raises no issues. But if your installation needs to use multiple clusters of AT servers, you need to make some adjustments. As a result of certain internal characteristics, it is not possible to deploy the RESTful facade on multiple clusters within the same domain. It *is* possible to deploy both the RESTful facade and the SOAP facade on a single cluster within the domain and to deploy only the SOAP facade on multiple other clusters within that same domain. In order to accomplish this, you need to use the normal mechanisms to undeploy the standard EAR files for the communication services (including the session manager) you are using on any additional clusters your installation requires and to deploy instead special equivalent EAR versions that contain only SOAP interfaces. The file names are identical to those of the standard files except that a **_soap** is appended to the end of the name. The following is a list of the SOAP-only EAR files that 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 **$BEAHOME/ocsg_5.1/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 this guide.

This procedure should be performed prior to production. It is recommended that you make the console-based changes (all except step 6) with only the Administration Server running. For step 6, you need to shut down all servers and re-start.

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. Carefully 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. Start up the Administration Server and install and deploy the SOAP-only EAR files to the non-REST cluster (WLNG_AT_Cluster).
10. Start the newly installed EAR files using the Administration console. If you forget to do this, their status will be "prepared".
11. Start the Managed Servers.

For more information on using the Administration Console to accomplish these tasks, click **Help** on the console screen.

Version Handling of Communication Services

Communication services are versioned and can be upgraded using in-production deployment.

The versioning is on EAR level, which means that all network protocol plug-ins for a given collection of application-facing interface are redeployed.

The version 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 Middleware Developing Applications for Oracle WebLogic Server* at

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13706/versioning.htm

Deploy and Undeploy Communication Services and plug-ins

Individual communication service are deployed and undeployed as EARs.

For a description of the different deployment options, see "Deploying Applications" in *Oracle Fusion Middleware Administrator's Guide* at

http://download.oracle.com/docs/cd/E15523_01/core.1111/e10105/deploy.htm

The EAR file names for each communication service and the JAR names for the network protocol plug-ins are listed a table in the "Properties for *communication service"* sections of the chapters in *Oracle Communications Services Gatekeeper Communication Service 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

To upgrade communication services, patches are applied using a patch tool. The patch tool operates on the EAR files for the communication services. The version of the plug-in is the same as the version of the service enabler it is packaged in, the versioning is on EAR file level.

Following is the typical process for applying patches to communication services.

1. The patch, in the form of a JAR file is provided.
2. The patch is applied to the original EAR file witch results in a new, patched, EAR. For example, if the original EAR version is 5.1, the version of the new EAR is 5.1_1. A sequence number should be suffixed to make each version unique.

3. The new EAR file is deployed and the old version is undeployed, using in-production redeployment, see "Hitless Upgrade Using Production Redeployment".
4. If another issue is detected for the same EAR a second patch JAR is provided.
5. The patch is applied to the already patched EAR and creates version 5.1_2.
6. The new patched EAR is deployed.

Alternatively, both patches can be applied at the same time. This will result in the same version as when applying them separately (version 5.1_2).

Patch Tool

Patches are applied using an Ant build file: *Middleware_Home*/**ocsg_5.1/server/bin/build.xml**

The correct `CLASSPATH` must be set up for the patch build file. Run **setWLSEnv.sh** or **setWLSEnv.cmd** located in *Middleware_Home*/**wlsserver_10.3/server/bin** to set up the environment.

There are two targets available in the build file:

- `patch` – Applies patches to an EAR file.
- `printpatches` – Displays the currently applied patches.

The syntax is:

```
ant [patch | printpatches] -Dsrc= -Ddest= -Dversion= -Dpatchdir= -Dpatchfiles=
```

Table 17–1 describes these arguments.

Table 17–1 Arguments to Patch Build File

Target/Argument	Description
patch	This target applies a patch.
printpatches	This target displays the currently applied patches. The information includes: ■ Manifest for the EAR ■ Manifest-Version ■ Bundle-Name ■ Created-By ■ Ant-Version ■ Weblogic-Application-Version ■ Bundle-Vendor ■ Bundle-Version ■ For each JAR in the EAR, patched class and ID of the patch is displayed.
src	The EAR file to which the patch is to be applied
dest	The EAR file that will contain the patch
version	The new version for the EAR. Must be different from the original version.
patchdir	Directory that holds the patches. Paths are relative or absolute.

Table 17–1 (Cont.) Arguments to Patch Build File

Target/Argument	Description
patchfiles	The JAR files that contain the patches. Wild cards are supported.

The patch tool detects conflicts if multiple patches try to patch the same file. In this situation, an error message is displayed and a combination (combo) patch must be used.

Examples

Table 17–2 contains some usage examples.

Table 17–2 Examples

Example	Command
View patch information for a single EAR file.	`ant printpatches -Dsrc=`*`original.ear`*
View patch information for all EARs in a specified directory.	`ant printpatches -Dsrc=/`*`path_to_ears`*

Apply a single patch to an EAR.	`ant patch -Dsrc=`*`original.ear`*` -Ddest=`*`patched.ear`*
	`-Dversion=5.1_1 -Dpatchdir=/`*`patches_directory`*
	`-Dpatchfiles=CR1234.jar`

Apply all patches located in the directory /patches.	`ant patch -Dsrc=`*`original.ear`*` -Ddest=`*`patched.ear`*
	`-Dversion=5.1_2 -Dpatchdir=/`*`patches_directory`*
	`-Dpatchfiles=*.jar`

Overview of Container Services and Configuration Files

Table 17–3 gives an overview of the Oracle Communications Services Gatekeeper configuration files.

Table 17–3 Oracle Communications Services Gatekeeper Configuration Files

File	Contains configuration for...

Domain_Home/**config/config.xml**	WebLogic Server. See "Domain Configuration Files" in *Oracle® Fusion Middleware Understanding Domain Configuration for Oracle WebLogic Server* at
	`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13716/config_files.htm`
	The following additions are specific to Oracle Communications Services Gatekeeper:
	...a <custom-resource>, with `<name>accesstier</name>`, that specifies the location of at.xml.
	...a <custom-resource>, with `<name>networktier</name>`, that specifies the location of nt.xml.
	...A set of <app-deployment> specific for the deployed communication services. See the relevant communication service in *Oracle Communications Services Gatekeeper Communication Service Guide*.
	Do not edit this file manually. 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 described in "Deployment Tools" in *Oracle® Fusion Middleware Deploying Applications to Oracle WebLogic Server* at
	`http://download.oracle.com/docs/cd/E15523_01/web.1111/e13702/understanding.htm`

Domain_Home/**config/custom/at.xml**	Oracle Communications 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**	Oracle Communications 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.

Container services

Table 17–4 describes the Oracle Communications Services Gatekeeper container services. The interfaces to these, together with common classes, are packaged in

Middleware_Home/**ocsg_5.1/server/lib/wlng/wlng.jar**

The container services are packaged in separate JARs, located in

Middleware_Home/**ocsg_5.1/server/lib/wlng/wlng.jar**

Table 17–4 Container Services

Container service	Summary
com.bea.wlcp.wlng.core.CoreServerService	Performs initial setup during the start-up sequence.
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 of Container Services

Patching of container services is done using SmartUpdate; see *WebLogic Server Installing Patches and Maintenance Packs* at

http://download.oracle.com/docs/cd/E12840_01/common/smartupdate/guide/

Patches to container services and WebLogic Server is done using rolling upgrades; see "Roadmap for Upgrading Your Application Environment" in *Oracle® Fusion Middleware Upgrade Guide for Oracle WebLogic Server* at

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13754/roadmap.htm

18

Hitless Upgrade Using Production Redeployment

Oracle Communications Services Gatekeeper supports seamless upgrades of Communication Services and Service Interceptors without service interruption. The upgrades are performed without disruptions in the traffic flow, without any need to restart servers, which means that the full processing capacity of the Oracle Communications Services Gatekeeper cluster is preserved during the hitless upgrade.

Hitless upgrades are performed using the production redeployment strategy, referred to in WebLogic Server as side-by-side redeployment. For more information, see "Deploying Applications" in *Oracle® Fusion Middleware Administrator's Guide* at:

http://download.oracle.com/docs/cd/E15523_01/core.1111/e10105/deploy.htm

Upgrades of container services provided by Oracle Communications Services Gatekeeper can not be done using production redeployment. Rolling upgrades should be used instead. See "WebLogic Server Rolling Upgrade" in *Oracle® Fusion Middleware Upgrade Guide for Oracle WebLogic Server* at:

http://download.oracle.com/docs/cd/E15523_01/web.1111/e13754/rolling_upgrade.htm

Production Redeployment Overview

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*. This means that it tracks any in-flight requests and makes sure that it does not get undeployed while having 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.

Production Redeployment Sequence

Following is a description of the production redeployment 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 the *Oracle Communications Services Gatekeeper System Administrator's Guide* chapter "Managing and Configuring the Plug-in Manager".

Requirements for Production Redeployment

For production redeployment to work:

- The old version of the plug-in must:
  - Perform protocol-specific cleanup of any connections or sessions towards 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 Scenarios for Production Redeployment

- 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.

- Parlay X 2.1 Short Messaging-Binary SMS/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 lifecycle 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.

- Communication services using 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.

- Communication services using SIP type plug-ins

  SIP-based Communication Services consist of an Oracle Communications Services Gatekeeper plug-ins and a SIP application. The Oracle Communications Services

Gatekeeper side plug-ins can be hitlessly upgraded the same way as other Oracle Communications Services Gatekeeper plug-ins.

Graceful retirement is not supported for SIP applications. For the SIP servlet part of the plug-ins, Oracle Communications Services Gatekeeper supports the same `-retiretimeout` option as WebLogic SIP Server.

- Extended Web Services Subscriber Profile/LDAP communication service

  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.

- 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.

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 *Communication service*” in this guide. See "Deployment Model for Communication Services and Container Services" for a description of how individual communication services are packaged.

Performing a Hitless Upgrade

Oracle Communications Services Gatekeeper uses the WebLogic Server mechanism for production redeployment.

Following is the syntax for using `weblogic.Deployer` to do the 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: R27.5.1-101-94136-1.5.1_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>
```

19

Managing and Configuring the Plug-in Manager

This chapter describes configuration and maintenance attributes and operations for the Plug-in Manager. It also provides a configuration workflow.

Introduction

Network protocol plug-ins consist of two parts: the plug-in service and the plug-in instance. The plug-in service is the deployable and updatable unit. It does not itself process any traffic. 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 service:

- Has a unique ID.
- Has a version.
- Is packaged in an EAR, together with other plug-ins that share the same set of application-facing interfaces.

A plug-in instance:

- Is instantiated from a plug-in service.
- 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.

Plug-in instances are created using the Plug-in Manager MBean.

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.

Execution and evaluation flow

The following section describes the execution and evaluation flow.

Application-initiated requests

When an application's request is processed, it is routed to the network protocol plug-in instance by the plug-in manager. The plug-in manager triggers a chain of service interceptors, where the request is evaluated and a given plug-in instance is selected based on:

- 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.
  - Address plan; that is the kind of network to which they are connected: for example E.164 or SIP.
- Usage policies based on SLA settings.
- Load balancing.
- Data from external sources: any custom data-lookups implemented as service interceptors.
- Routing logic

When the decision has been made, the request is forwarded to the selected plug-in.

Routing logic is expressed in XML, and is evaluated by means of service interceptors.

Refer to the section in this guide for each communication service for the plug-in service ID and plug-in type under which individual plug-ins are registered.

The following interceptors use data configured using the Plug-in Manager:

- CreatePluginList
- RemoveInvalidRoute

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.

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 towards the network node.

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.

The logical operators are XML elements, as described in Table 19–1.

Table 19–1 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 19–2.

Table 19–2 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 "How address ranges are specified when setting up routes" for examples of regular expressions. The element has the optional attribute `defaultResult`. It is a Boolean attribute that affects the end 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.
	Examples:
	`<method>sendSms</method>`
	`<method>makeACall</method>`

Table 19–2 (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 Operation: getServiceInfo for a list of parameter names. - `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 end 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. 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 end 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.
	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 19–1 matches requests sent to an address starting with `tel:123`. If the request does not contain an address this route does not match.

Example 19–1 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 19–2 matches any address and also matches any request that does not contain an address.

Example 19–2 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 19–3 matches requests sent from the service provider with the ID `sp1`, and the application with the ID `app1`.

Example 19–3 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 19–4 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 19–4 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 19–5 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>
```

How address ranges are specified when setting up 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, in the expression "^46.*" the "." 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.

Plug-in Routing Logic and Plug-in Routes

Plug-in routing logic is an extension the plug-in routes and Operation: setRouteConfig and Operation: getRouteConfig replaces Operation: addRoute and Operation: removeRoute.

Plug-in routes added using Operation: addRoute, are converted into plug-in routing logic where route is added as a logical OR given that the route that is being modified only contain `<or>` and `<destAddress defaultResult="true">` elements.

Configuration Workflow

Configuring the Plug-in Manager can be divided into:

- Configuration of the general behavior of the plug-in manager: see "Configuring the Plug-in Manager".
- Administration of routes, which is tightly coupled to the configuration of the individual communication services: see "Configuring the Plug-in Manager".

Configuring the Plug-in Manager

Following is an outline for configuring the Plug-in Manager:

1. Specify whether or not to use policy-based routing in Attribute: PolicyBasedRouting. 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 Attribute: ForceConnectInResuming.

Creating a Plug-in instance

Following is an outline for creating an instance of a plug-in service:

1. Find the plug-in service ID for the service you wish 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 Operation: listPluginServices.
2. Use Operation: createPluginInstance 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 Operation: destroyPluginInstance.

Administration of Plug-in Routing Logic and Node IDs

The administration of routes uses the following operations:

- To add new routing logic: Operation: setRouteConfig
- To change or remove routing logic: Operation: getRouteConfig and Operation: setRouteConfig
- To list all existing routes: Operation: listRoutes
- To get information about a specific plug-in instance: Operation: getPluginServiceInfo
- To get a list of all registered plug-in instances: Operation: listPluginInstances
- To get a list of all registered plug-in services: Operation: listPluginServices
- To define the node ID: Operation: 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.

Reference: Attributes and Operations for PluginManager

Managed object: Container Services->PluginManager

MBean: com.bea.wlcp.wlng.plugin.PluginManagerMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: ForceConnectInResuming
- Attribute: PolicyBasedRouting
- Attribute: supportBulkRequest
- Operation: addRoute

- Operation: createPluginInstance
- Operation: destroyPluginInstance
- Operation: getPluginInstanceInfo
- Operation: getPluginNodeId
- Operation: getPluginServiceInfo
- Operation: getRouteConfig
- Operation: listPluginInstances
- Operation: listPluginServices
- Operation: listRoutes
- Operation: listServiceTypes
- Operation: removeRoute
- Operation: setPluginNodeId
- Operation: setRouteConfig

Attribute: ForceConnectInResuming

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if a network protocol plug-in service is allowed to transition to state ACTIVE if it fails to establish a connection with the underlying network node during deployment or re-deployment. This assures that a new version of the network protocol plug-in service is activated only when it is certain that it can accept traffic.

If enabled, all plug-in services packaged in the EAR that is being updated must become active before any traffic is routed to the plug-in instances derived from the new version. If any plug-in instance fails to connect to the underlying network, the new version does not become active.

Use:

- `true` to check if a connection can be established before accepting traffic.
- `false` to not perform this check.

Attribute: PolicyBasedRouting

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether policy based routing should be used.

Use:

- `true` to enable policy-based routing
- `false` to disable policy-based routing

Note: Policy based routing must be enabled to enforce node SLAs.

Attribute: supportBulkRequest

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether bulk request is supported. If set to true, SMS requests with multiple recipients are split into multiple individually-addressed SMS requests.

Customized message content for each address included in the submission is supported when supportBulkRequest is set to true and a DifferentContentForSingleAddressInBulk xparam, is also set to true and added to the request.

Use:

- `true` to enable bulk request
- `false` to disable bulk request

Operation: addRoute

Scope: Cluster

Deprecated. Use Operation: setRouteConfig.

Adds a new plug-in route. A route is identified by the plug-in instance ID and a match pattern.

Signature:

```
addRoute(PluginInstanceId: String, AddressExpression: String)
```

Table 19–3 describes these parameters.

Table 19–3 addRoute

Parameter	Description
PluginInstanceId	ID of the plug-in instance that is the target of the route. A list of plug-in instance IDs is displayed using the Operation: listPluginInstances.
AddressExpression	The pattern to be used as a matching criteria. See "How address ranges are specified when setting up routes".

Operation: createPluginInstance

Scope: Cluster

Creates a new instance of a specific plug-in service.

Signature:

```
createPluginInstance(PluginServiceId: String, PluginInstanceId: String)
```

Table 19–4 describes these parameters.

Table 19–4 createPluginInstance

Parameter	Description
PluginServiceId	ID of the plug-in service to instantiate. See "Operation: listPluginServices" for a list of plug-in service IDs.

Table 19–4 (Cont.) createPluginInstance

Parameter	Description
PluginInstanceId	ID of the plug-in instance to be created. The ID must be unique among all instances. All existing instances can be displayed using Operation: listPluginInstances.

Operation: destroyPluginInstance

Scope: Cluster

Destroys an instance of a specific plug-in instance. All routes associated with the instance are kept.

Signature:

```
destroyPluginInstance(PluginServiceId: String, PluginInstanceId: String)
```

Table 19–5 describes these parameters.

Table 19–5 destroyPluginInstance

Parameter	Description
PluginServiceId	ID of the plug-in service that has a plug-in instance that is to be destroyed
PluginInstanceId	ID of the plug-in instance to destroy

Operation: getPluginInstanceInfo

Scope: Cluster

Gets information about a specific plug-in instance. The information includes:

- The node ID, if the plug-in instance is assigned a node ID using Operation: setPluginNodeId.
- A list of application-facing interfaces registered with the plug-in manager.
- A list of network-facing interfaces registered with the plug-in manager.
- Whether the plug-in instance is connected to the network node (true if connected).
- A list of all configured routes.

Signature:

```
getPluginInstanceInfo(PluginInstanceId: String)
```

Table 19–6 describes this parameter.

Table 19–6 getPluginInstanceInfo

Parameter	Description
PluginInstanceId	ID of the plug-in instance whose information is to be retrieved.

Operation: getPluginNodeId

Scope: Cluster

Displays which plug-in node ID a plug-in instance is associated with, if any.

Signature:

```
getPluginNodeId(PluginInstanceId: String)
```

Table 19–7 describes this parameter.

Table 19–7 getPluginNodeId

Parameter	Description
PluginInstanceId	ID of the plug-in

Operation: getPluginServiceInfo

Scope: Cluster

Displays information about a plug-in service. The information includes:

- The service type.
- A list of address schemes the plug-in service supports.
- The protocol it uses towards the network node.
- A list of plug-in instances created based on the plug-in service.

Signature:

```
getPluginServiceInfo(PluginServiceId: String)
```

Table 19–8 describes this parameter.

Table 19–8 getPluginServiceInfo

Parameter	Description
PluginServiceId	ID of the plug-in service

Operation: getRouteConfig

Scope: Cluster

Gets the plug-in routing logic XML for a given plug-in instance.

Signature:

```
getRouteConfig(PluginInstanceId :String)
```

Table 19–9 describes this parameter.

Table 19–9 getRouteConfig

Parameter	Description
PluginInstanceId	The ID of the plug-in instance whose routing configuration is to be retrieved

Operation: getServiceInfo

Scope: Cluster

Displays information about a service type. The information includes:

- A list of interface names for the service type.
- A list of method names for each interface.

- A list of argument names for each method.
- The name of the return type for each method.

This list is useful when setting up SLAs.

Signature:

```
getServiceInfo(Type: String)
```

Table 19–10 describes this parameter.

Table 19–10 getServiceInfo

Parameter	Description
Type	ID of the service type. See "Operation: listServiceTypes".

Operation: listPluginInstances

Scope: Cluster

Displays a list of plug-in instance IDs.

The list is rendered as:

Plug-in instance ID#JEE application name#*version of JEE application*

The plug-in instance ID is the part of each entry up to the first #.

Signature:

```
listPluginInstances()
```

Operation: listPluginServices

Scope: Cluster

Displays a list of plug-in service IDs. A plug-in service ID uniquely identifies a plug-in service.

Signature:

```
listPluginServices()
```

Operation: listRoutes

Scope: Cluster

Displays a list of all registered routes, including the routing logic expressed an XML.

Signature:

```
listRoutes()
```

Operation: listServiceTypes

Scope: Cluster

Displays a list of service types. The service type defines a collection of plug-in services exposing a specific functionality in the network.

For example: SMS or MMS.

Signature:

```
listServiceTypes()
```

Operation: removeRoute

Scope: Cluster

Deprecated. Use Operation: setRouteConfig.

Removes a route. The route is identified by the ID of the plug-in instance and the matching pattern.

Signature:

```
removeRoute(PluginInstanceId: String, AddressExpression: String)
```

Table 19–11 describes these parameters.

Table 19–11 removeRoute

Parameter	Description
PluginInstanceId	ID of the plug-in
AddressExpression	The pattern used as a matching criteria. See "How address ranges are specified when setting up routes".

Operation: setRouteConfig

Scope: Cluster

Loads plug-in routing logic XML for a plug-in instance. Existing routing logic is overwritten, use Operation: getRouteConfig to get the existing routing logic.

Replaces Operation: addRoute.

Signature:

```
setRouteConfig(PluginInstanceId :String, xmlConfig: String)
```

Table 19–12 describes these parameters.

Table 19–12 setRouteConfig

Parameter	Description
PluginInstanceId	The plug-in instance ID to load the routing configuration for.
xmlConfig	The plug-in routing logic. Expressed as XML, see "Plug-in Routing Logic".

Operation: setPluginNodeId

Scope: Cluster

Assigns a node ID to a plug-in instance.

Signature:

```
setPluginNodeId(PluginInstanceId :String, nodeId: String)
```

Table 19–13 describes these parameters.

Table 19–13 setPluginNodeId

Parameter	Description
PluginInstanceId	The plug-in instance ID.

Table 19–13 (Cont.) setPluginNodeId

Parameter	Description
nodeId	ID to be used as a node ID. This ID is used when enforcing node SLAs.

Operation: listInterceptors

Scope: Domain

The **listInterceptors** operation retrieves a list of all enabled interceptors in Services Gatekeeper.

Signature

```
listInterceptors(InterceptionPoint: String)
```

Table 19–14 describes these parameters.

Table 19–14 listInterceptors

Parameter	Description
InterceptionPoint	The Services Gatekeeper interface where the interceptor(s) are applied (MT_NORTH, MT_SOUTH, MO_NORTH or MO_SOUTH).

Operation: retrieveInterceptorConfiguration

Scope: Domain

The **retrieveIneterceptorConfiguraiton** operation retrieves the active interceptor rule configuration in Services Gatekeeper.

Signature

```
retrieveInterceptorConfiguration()
```

Operation: updateInterceptorConfiguration

Scope: Domain

The **updateInterceptorConfiguration** operation updates the Services Gatekeeper interceptor rule configuration.

Signature

```
updateInterceptorConfiguration(Sla : String)
```

Table 19–15 describes these parameters.

Table 19–15 updateInterceptorConfiguration

Parameter	Description
Sla	The contents on an **interceptorRule.xml** file. For an example, see the information on service interceptors in the *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide*.

20

Managing and Configuring the Tier Routing Manager

This chapter describes how to configure and maintain attributes and operations for the Tier Routing Manager. It also provides a workflow for the configuration.

Introduction

Applications can interact with these Service Facades when using Services Gatekeeper communication services:

- SOAP
- SOA
- RESTful
- Native

All Service Facades use Service Enablers to connect to the telecom network.

Application-initiated requests are automatically routed from any given 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 20–1 for examples of endpoint expressions.

Table 20–1 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.

Configuration Workflow

The administration of routes uses the following operations:

- To add a new tier route, consult Operation: addTierRoute
- To remove a tier route, consult Operation: removeTierRoute
- To list all existing tier routes, consult Operation: listTierRoutes

Reference: Attributes and Operations for TierRoutingManager

Managed object: Container Services->TierRoutingManagerMBean

MBean: com.bea.wlcp.wlng.tier_routing.TierRoutingManagerMBean

Following is a list of attributes and operations for configuration and maintenance:

- Operation: addTierRoute
- Operation: listTierRoutes
- Operation: removeTierRoute

Operation: addTierRoute

Scope: Cluster

Adds a new tier route. A route is identified by an index.

Signature:

```
addTierRoute(endpointExpression : String, clusterName: String, index: int)
```

Table 20–2 addTierRoute

Parameter	Description
endpointExpression	The pattern to be used as a matching criteria for the callback URL for network-triggered operations. Each Service Facade adds its own signature to the pattern. See Table 20–1 for examples of routes. The pattern is a regular expression.
clusterName	The name of the cluster to pass the request to if the endpointExpression matches.
index	The index of the tier route. The first matching route will be used so the order of the routes are important if a URL can match multiple routes. The new entry will shift any existing entries one index higher. To insert entry first in the list, use index 0 (zero). To insert entry last in the list, use the number of list entries (the size of the list).

Operation: listTierRoutes

Scope: Cluster

Displays the list of tier routes. The list includes:

- index
- endpoint expression
- cluster name

Signature:

```
listTierRoutes()
```

Operation: removeTierRoute

Scope: Cluster

Removes a tier route. The route is identified by the index. After the update, the index of all tier routes with a higher index than the removed are decreased by one.

Signature:

```
removeTierRoute(index: int)
```

Table 20–3 describes this parameter.

Table 20–3 removeTierRoute

Parameter	Description
index	Index if the tier route to remove.

21

SOAP2SOAP Communication Services

This chapter describes how to generate, deploy and configure SOAP2SOAP communication services.

About SOAP2SOAP Communication Services

You can create SOAP2SOAP communication Services using the Platform Development Studio or directly from the Services Gatekeeper Administration Console

Based on a set of service WSDLs and callback WSDLs, a SOAP2SOAP communication service acts as a proxy service and provides the functionality provided by Services Gatekeeper container services, such as:

- SLA enforcement
- EDR, CDR, and alarms
- Authentication, access control, and accounting

SOAP2SOAP plug-in services can be instantiated using the plug-in manager. The plug-in instances process traffic and connect to individual network nodes. The instances are managed independently of each other.

For application-initiated requests, all requests are routed to the network node defined for the plug-in instance.

For network-triggered requests, the network node should distinguish the application instance the request is targeted to by adding the application instance ID to the URL:

```
http://IP_Address_of_AT_server:port/context-root_nt/plug-in instance ID_
nt/application_instance_ID
```

The endpoint at which the application instance has implemented the Web Service is provisioned using "Operation: addApplicationEndPoint": see "Provisioning Workflow for SOAP2SOAP Communication Services" for a list of related operations.

If the number of HTTP requests that time-out during a certain time-period exceeds the maximum allowed, the plug-in instance is taken out of operation for a configurable time-period. The HTTP time-out behavior is valid for request between the plug-in instance and the network node. The purpose of this feature is to prevent network nodes that are malfunctioning from incoming requests.

Generating and Deploying SOAP2SOAP Communication Services

SOAP2SOAP communication services can be generated and deployed using the Administration console:

- Generate a Communication Service
- Deploy a Communication Service

See "Generated Artifacts for the Communication Service" for a description of the generated artifacts.

Generate a Communication Service

To generate a SOAP2SOAP communication service from the Administration console, open the SOAP-SOAP Generation page by selecting OCSG > SOAP-SOAP from the Domain Structure group and enter configuration properties for the communication service using the information in to Table 21–2.

Table 21–1 SOAP-SOAP Communication Service Generation Fields

Field	Description
Name	Identifier that ties together a collection of Web Services. Will be a part of the names of the generated war and jar files and the package names for the for the communication service. Following is a list of the generated EARs and the WARs and JARs: wlng_at_*Communication service name*.ear: ■ *Name*.war ■ *Name*_callback.jar wlng_nt_*Communication service name*.ear: ■ *Name*.war ■ *Name*_service.jar ■ *Name*_soap_plugin.jar
Version	The version to be used in META-INF/MANIFEST.MF in the EARs for the communication service.
ServiceType	Defines the service type. Used in SLAs, EDRs, statistics, and so on. The service type to use in the SLAs is: com.bea.wlcp.wlng.soap2soap.*ServiceType*.plugin.*interface_name from_WSDL* Example: SmsServiceType
ContextPath	The context path for the Web Service. Example: myService
Service WSDLs	Enter the name of and path to each WSDL file that includes the service definition to be implemented by the new communication service. Use one file per line. The files must reside on a file system that can be reached by the Administration server. Examples: D:\standards\ParlayX2.1\wsdl\parlayx_sms_send_service_2_2.wsdl D:\standards\ParlayX2.1\wsdl\parlayx_sms_notification_manager_service_2_3.wsdl D:\standards\ParlayX2.1\wsdl\parlayx_sms_receive_service_2_2.wsdl

Table 21–1 (Cont.) SOAP-SOAP Communication Service Generation Fields

Field	Description
Callback WSDLs	Enter the name of and path to each WSDL file that includes the callback service definition to be used by the new communication service in sending information to the service provider's application. Use one file per line. The files must reside on a file system that can be reached by the Administration server. Example: D:\standards\ParlayX2.1\wsdl\parlayx_sms_notification_service_2_2.wsdl

On successful generation of the communication service, the deployment screen is displayed: see "Deploy a Communication Service".

You can also generate a SOAP2SOAP communication service by using the Eclipse wizard provided with the Platform Development Studio. See "Using the Eclipse Wizard" in the *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide*.

Deploy a Communication Service

After a SOAP2SOAP communication service has been generated using the Administration Console (see "Generate a Communication Service"), the deployment screen is displayed.

When deploying directly from the Administrating console, complete the entry fields described in Table 21–2 and click the **Deploy SOAP2SOAP** button. In this case, the communication service is deployed to the same domain as where it was generated.

Table 21–2 SOAP-SOAP Communication Service Deployment Fields

Field	Description
Name	Name of the communication service. Same as the name entered in the Name field in SOAP2SOAP Communication Service screen. Do not change this.
Version	The version use when deploying the communication services.
UserName	User name for an Oracle Communications Services Gatekeeper administrative user.
Password	Password that corresponds to the user name.

The generated communication service can also be deployed using WebLogic tools: see "Generated Artifacts for the Communication Service".

After the communication service has been deployed, create one or more plug-in instances. See "Managing and Configuring SOAP2SOAP Communication Services" for more information.

Generated Artifacts for the Communication Service

When a SOAP2SOAP communication service is generated, the following directory structure is created in the directory *Domain_Home*/**soap2soap** on the Administration Server.

Example 21–1 Directory Structure from Generating SOAP2SOAP Communication Services

```
+- <Communication service name>_<version>/
+- build.properties
+- build.xml
+- common.xml
+- <Communication service name>/
+- common/+- resources/
+- enabler/
+- facade/
+- handlerconfig.xml
+- dist/
+- com.bea.wlcp.wlng.soap2soap.<service type>.store_<version>.jar
+- wlng_at_<Communication service name>.ear
+- wlng_nt_<Communication service name>.ear
+- plugins/
|+- soap/
 +- tmp/
```

All Java source files, build files, configuration files, and deployable artifacts are created under the directory *Domain_Home*/**soap2soap**. The source files are there mainly for reference. The only files that are necessary to deploy the communication service are:

- wlng_at_*communication_service_name*.ear
- wlng_nt_*communication_service_name*.ear

wlng_at_*communication_service_name*.ear should be deployed in the Access Tier server.

wlng_nt_*communication_service_name*.ear should be deployed in the Network Tier server.

The generated communication service can be deployed as a part of the generation process, as described above, or using standard WebLogic Server tools.

See "Deploy and Undeploy Communication Services and plug-ins" for information on how to deploy the files using WebLogic Server tools.

Managing and Configuring SOAP2SOAP Communication Services

This section contains a description of the configuration attributes and operations available for SOAP2SOAP-type of plug-in instances.

Table 21–3 Task Overview

To see a	Refer to
Detailed list of necessary for managing and configuring a plug-in instance	Properties for SOAP2SOAP Plug-ins
Configuration workflow	Configuration Workflow for SOAP2SOAP Plug-ins
List of operations and attributes related to management and provisioning	Provisioning Workflow for SOAP2SOAP Communication Services

Table 21–3 (Cont.) Task Overview

To see a	Refer to
Reference of management attributes and operations	Reference: Attributes and Operations for SOAP2SOAP Plug-ins

Properties for SOAP2SOAP Plug-ins

Table 21–4 Properties

Property	Description
Managed object in Administration Console	*domain name* > OCSG > *server name* > Communication Services > *plug-in instance ID*

MBean	Domain=com.bea.wlcp.wlng
	Name=wlngt
	InstanceName is same as plug-in instance ID
	Type=com.bea.wlcp.wlng.httpproxy.management.HTTPProxyMBean

Network protocol plug-in service ID	Defined when generating the SOAP2SOAP plug-in using the Platform Development Studio. When generated fm the Administration console, the ID is *Name*_soap_plugin.
Network protocol plug-in instance ID	The ID assigned when the plug-in instance is created: see "Managing and Configuring the Plug-in Manager".
Supported Address Scheme	Defined when generating the SOAP2SOAP plug-in using the Platform Development Studio. When generated fm the Administration console, the address scheme is always an empty string.

Application-facing interfaces	Derived from the package name of the network protocol plug-in, assigned when the plug-in was generated, and the name of the interface as defined in the WSDL.
	For application-initiated request:
	`package name.plugin.interface namePlugin`
	For network-triggered requests:
	`package name.callback.interface nameCallback`
	See "Managing and Configuring the Plug-in Manager" for information on how to list the interfaces.

Service type	Given when the plug-in was generated using the Platform Development Studio or the administration console.
Exposes to the service communication layer a Java representation of:	Depends on the WSDLs used.
Interfaces with the network nodes using:	The same protocol as exposed by the application-facing interfaces.

Table 21–4 (Cont.) Properties

Property	Description
Deployment artifacts	*communication service identifier_protocol*_plugin.jar, *communication service identifier*_service.jar and *communication service identifier*_ callback.war, packaged in wlng_nt_*communication service*.ear *communication service identifier*_callback.jar and *communication service identifier*.war, packaged in wlng_at_*communication service identifier*.ear *communication service identifier* was given in the Eclipse wizard or the Administration console when the communication service was generated. *protocol* is configurable when using the Eclipse wizard. If the communication service was generated using the Administration console, it is always `soap`.

Configuration Workflow for SOAP2SOAP Plug-ins

Following is an outline for configuring the plug-in using the Administration Console:

1. Create one or more instances of the plug-in service: see "Managing and Configuring the Plug-in Manager". Use the plug-in service ID as detailed in "Properties for SOAP2SOAP Plug-ins".
2. Using the Administration Console or an MBean browser, select the MBean for the plug-in instance. The MBean display name is the same as the plug-in instance ID assigned when the plug-in instance was created.
3. Configure the authentication credentials to be used by the plug-in towards the network node:
   - Attribute: UserName
   - Attribute: Password
4. Specify which authentication method to use towards the network node:
   - Attribute: HttpBasicAuthentication
   - Attribute: UserNameTokenAuthentication
5. Specify the URL of the network node to connect to:
   - Attribute: UserNameTokenAuthentication
6. Specify the HTTP time-out behavior:
   - Attribute: HttpTimeoutPeriod
   - Attribute: HttpTimeoutThreshold
   - Attribute: HttpWaitTime
   - Attribute: ReactivateTime
7. Specify heartbeat behavior; see Configuring Heartbeats.
8. Set up the routing rules to the plug-in instance: see "Configuring the Plug-in Manager". Use the plug-in instance ID and address schemes detailed in "Properties for SOAP2SOAP Plug-ins".

> **Note:** If the SOAP2SOAP plug-in is the only plug-in for the service enabler, routing based on destination address is not applicable. All requests will be routed to the plug-in instance. If there are non-SOAP2SOAP plug-ins in the service enabler, the routing applies.

9. Provide the administrator of the network node server with the URL to which it should deliver network-triggered requests. The default URL is

   ```
   http://IP Address:port/context-root_nt/plug-in instance ID/application instance
   ID
   ```

   The default *context-root* is the communication service ID, as specified when the communication service was generated.

   The *plug-in instance ID* is the ID of the plug-in instance. The network node is assumed to specify which application instance the request is targeted to by adding the application instance ID as the suffix in the URL.

10. If required, create and load a node SLA. For details see "Defining Global Node and Service Provider Group Node SLAs" and "Managing SLAs" in the *Oracle Communications Services Gatekeeper Accounts and SLAs Guide*.

Continue with the provisioning of service provider accounts and application accounts.

Provisioning Workflow for SOAP2SOAP Communication Services

For each application that uses SOAP2SOAP communication services and supports network-triggered requests, set up a mapping between the application instance ID and the URL for the Web service that the application instance implements. Use the following operations to manage the callback URLs for the application instances:

- Operation: addApplicationEndPoint
- Operation: getApplicationEndPoint
- Operation: listApplicationEndPoints
- Operation: removeApplicationEndPoint

Reference: Attributes and Operations for SOAP2SOAP Plug-ins

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: HttpBasicAuthentication
- Attribute: HttpTimeoutPeriod
- Attribute: HttpTimeoutThreshold
- Attribute: HttpWaitTime
- Attribute: NetworkEndpoint
- Attribute: Password
- Attribute: UserName
- Attribute: ReactivateTime
- Attribute: UserNameTokenAuthentication
- Operation: addApplicationEndPoint
- Operation: getApplicationEndPoint

- Operation: listApplicationEndPoints
- Operation: removeApplicationEndPoint

Attribute: HttpBasicAuthentication

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether HTTP Basic Authentication shall be used when authenticating with the network node. Enter:

- `true` to use HTTP Basic Authentication
- `false` otherwise.

Attribute: HttpTimeoutPeriod

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies a time period during which a number of HTTP time-outs are counted.

Attribute: HttpTimeoutThreshold

Scope: Cluster

Unit: Not applicable

Format: Integer

Specifies the number of HTTP time-outs allowed during a time-period. The time-period is specified in Attribute: HttpTimeoutPeriod.

If the number of time-outs are exceed, the plug-in instance transfers to state disconnected, which means that it does not accept new requests. It stays disconnected during the time-period defined in Attribute: ReactivateTime.

Attribute: HttpWaitTime

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the HTTP time-out value. If this value is exceeded, the request is considered to be lost and the counter that tracks the number of requests that encountered HTTP time-outs is increased.

Attribute: NetworkEndpoint

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the URL to the network node.

Attribute: Password

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the password to use when connecting to the network node.

Attribute: ReactivateTime

Scope: Cluster

Unit: Seconds

Format: Integer

Specifies the time a plug-in instance is kept in state disconnected when the number of HTTP time-outs have exceeded the maximum number, specified in Attribute: HttpTimeoutThreshold, over a time-period, specified in Attribute: HttpTimeoutPeriod.

Attribute: UserName

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the user name to use when connecting to the network node.

Attribute: UserNameTokenAuthentication

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies whether Web Services Security UserName Token is used to authenticate with the network node.

- `true` to use UserName Token Authentication
- `false` otherwise.

Operation: addApplicationEndPoint

Scope: Cluster

Adds the URL to which network-triggered requests per application instance should be forwarded. The application is assumed to implement the callback web service on this URL.

Signature:

```
addApplicationEndPoint(AppinstanceId: String, CallbackUrl: String)
```

Table 21–5 addApplicationEndPoint

Parameter	Description
AppinstanceId	The application instance ID associated with the callback URL.

Table 21–5 (Cont.) addApplicationEndPoint

Parameter	Description
CallbackUrl	The URL to the application instance’s implementation of the callback.

Operation: getApplicationEndPoint

Scope: Cluster

Displays the URL to which network-triggered requests for a given application instance should be forwarded.

Signature:

```
getApplicationEndPoint(AppinstanceId: String)
```

Table 21–6 getApplicationEndPoint

Parameter	Description
AppinstanceId	The application instance ID associated with the callback URL.

Operation: listApplicationEndPoints

Scope: Cluster

Displays a list of all registered callback URLs.

Signature:

```
listApplicationEndPoints()
```

Operation: removeApplicationEndPoint

Scope: Cluster

Removes the URL to which the network-triggered requests for a given application instance are forwarded.

Signature:

```
removeApplicationEndPoint(AppinstanceId: String)
```

Table 21–7 removeApplicationEndPoint

Parameter	Description
AppinstanceId	The application instance ID associated with the callback URL.

22 Configuring Heartbeats

This chapter describes how to configure heartbeats for HTTP-based stateless network protocol plug-ins.

Introduction

The heartbeat functionality performs heartbeat checks towards HTTP-based network nodes on behalf of a plug-in. When a heartbeat fails, the plug-in is set to state INACTIVE. The heartbeat functionality will continue 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

Configuration and Management

The heartbeat OAM 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:
  - 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 need to 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.

1. Specify if heartbeats should be enabled for the associated plug-in in Attribute: Enabled.
2. If heartbeats are enabled, define:

   Attribute: ContentMatch

   Attribute: Interval

   Attribute: NetworkServiceUrl

Reference: Attributes and Operations for HeartbeatConfiguration

Managed object: HeartbeatConfiguration

MBean: om.bea.wlcp.wlng.heartbeat.management.HeartbeatMBean

Following is a list of attributes and operations for configuration:

- Attribute: ContentMatch
- Attribute: Enabled
- Attribute: HeartbeatMethod
- Attribute: Interval
- Attribute: NetworkServiceUrl
- Attribute: PostContent
- Attribute: ResponseCodeToMatch
- Attribute: Status (r)
- Operation: addHTTPHeader
- Operation: listHTTPHeaders
- Operation: removeHTTPHeader

Attribute: ContentMatch

Scope: Cluster

Unit: Not applicable

Format: String

The content for matching the returned heartbeats. If the returned string differs from this, the heartbeat is assumed to have failed. Leave this empty to match any content.

Attribute: Enabled

Scope: Cluster

Unit: Not applicable

Format: Boolean

Specifies if the heartbeat functionality is applied for the plug-in.

If set to `true` when the server is stopped, remains true so that hearbeats are automatically enabled after a server restart.

Attribute: HeartbeatMethod

Scope: Cluster

Unit: Not applicable

Format: String

Valid values are "GET" and "POST". The value determines whether to use HTTP GET or POST method for heartbeats.

Content can be added to the HTTP request when the value is "POST".

Content matching is performed only when the value is "GET."

Response code matching is performed in both cases.

The default is "GET".

Attribute: Interval

Scope: Cluster

Unit: seconds

Format: integer

Specifies the time interval between heartbeats.

Attribute: NetworkServiceUrl

Scope: Cluster

Unit: Not applicable

Format: String

Specifies the URL to which heartbeats are sent.

Attribute: PostContent

Scope: Cluster

Unit: Not applicable

Format: String

The content to post in the HTTP request. Used only when the `HeartbeatMethod` is POST. See Attribute: HeartbeatMethod.

Attribute: ResponseCodeToMatch

Scope: Cluster

Unit: Not applicable

Format: integer

The response code to match.

Response code matching is performed in all cases, whether the `HeartbeatMethod` attribute is GET or POST. See Attribute: HeartbeatMethod.

The default is 200.

Attribute: Status (r)

Scope: Cluster

Unit: Not applicable

Format: Boolean

Read-only.

If `true`, there was a successful response to the last heartbeat. If the previous status was `false`, the plug-in state is altered from INACTIVE to ACTIVE.

If `false`, there was no response or an incorrect response to the last heartbeat. In this case, the associated plug-in is set to state INACTIVE and an alarm is generated.

Operation: addHTTPHeader

Scope: Cluster

Adds an HTTP header to all POST and GET heartbeat requests.

If a header of the specified name already exists, the value passed in this operation replaces the previous value.

Signature:

```
addHTTPHeader(Name: String, Value: String)
```

Table 22–1 addHTTPHeader

Parameter	Description
Name	Name of header being added.
Value	Contents of header being added

Operation: listHTTPHeaders

Scope: Cluster

Lists all the headers included in heartbeat requests.

Signature:

```
listHTTPHeaders()
```

Operation: removeHTTPHeader

Scope: Cluster

Removes an HTTP header from all heartbeat requests.

```
removeHTTPHeader((Name: String)
```

Table 22–2 removeHTTPHeader

Parameter	Description
Name	Name of header being removed.

23

Managing and Configuring Connection Information

This chapter describes how to create and store connection and credential mappings in the Services Gatekeeper Connection Information Manager. The Connection Information Manager is used by the native UCP plug-ins.

Introduction

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 connection information is stored in a ConnectInfo object.

The ConnectInfo object:

- Stores the remote address to which the plug-in connects
- Stores 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 Operation: getConnectInfo for details about the ConnectInfo object.

Configuration and Management

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 > 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 the MBean is accessed.

If you are using different plug-ins that use the Connection Information Manager, you need to configure the Connection Information Manager for each plug-in.

Use the following operations to configure and manage the connection information:

- Operation: createOrUpdateLocalHostAddress
- Operation: createOrUpdateRemoteHostAddress
- Operation: getConnectInfo
- Operation: removeConnectInfo
- Operation: removeLocalHostAddress

Use the following operations to configure and manage the credential map:

- Operation: addXParamToCredentialEntry
- Operation: createOrUpdateCredentialMap
- Operation: createOrUpdateUserPasswordCredentialEntry
- Operation: listAllCredentialEntries
- Operation: removeCredentialEntry
- Operation: removeCredentialMap

Use the following operations to configure and manage the listen information:

- Operation: createOrUpdateListenAddress
- Operation: getAllListenAddress
- Operation: removeListenAddress
- Operation: getListenAddressForCurrentServer

Reference: Attributes and Operations for Connection Information Manager

Managed object: ConnectInfoManager

MBean: oracle.ocsg.plugin.connection.management.ConnectInfoManagerMBean

Following is a list of attributes and operations for configuration:

- Attribute: ValidationEnabled
- Operation: addXParamToCredentialEntry
- Operation: createOrUpdateCredentialMap
- Operation: createOrUpdateListenAddress
- Operation: createOrUpdateLocalHostAddress
- Operation: createOrUpdateRemoteHostAddress
- Operation: createOrUpdateUserPasswordCredentialEntry
- Operation: getAllListenAddress
- Operation: getConnectInfo
- Operation: getListenAddressForCurrentServer
- Operation: listAllCredentialEntries
- Operation: removeConnectInfo
- Operation: removeCredentialEntry
- Operation: removeCredentialMap

- Operation: removeListenAddress
- Operation: removeLocalHostAddress
- Operation: removeLocalHostAddress

Attribute: ValidationEnabled

Scope: Cluster

Unit: Not applicable

Format: Boolean

If set to `true`, the Plug-in Manager validates the plug-in instance ID before the Connection Information Manager creates an entry for it in the credential map.

It also validates the server names and cluster names targets passed in the `targets` parameters to the createOrUpdateListenAddress and createOrUpdateLocalHostAddress operations.

See Operation: createOrUpdateListenAddress and Operation: createOrUpdateLocalHostAddress.

The default is `true`.

Operation: addXParamToCredentialEntry

Scope: Cluster

Adds additional parameter values to the credential map associated with the `credentialId`.

Use this operation to add any name-value pair to the operation. For example, native UCP can use this operation to provide connection-specific windowing and heartbeat parameters.

Signature:

```
addXParamToCredentialEntry(credentialId: String, xParamName: String, xParamValue:
String)
```

Table 23–1 addXParamToCredentialEntry

Parameter	Description
credentialId	Unique identifier for the credential in the configuration; created by a previous call to "Operation: createOrUpdateCredentialMap".
xParamName	User-defined name for the parameter
xParamValue	Value of the parameter

Operation: createOrUpdateCredentialMap

Scope: Cluster

Creates a new entry or updates an existing entry in the credential map associated with the `pluginInstanceId`.

This operation creates the association between a plug-in instance, an application instance, and a credential ID.

For credential maps configured for use by native UCP, the `appId` must be a numeric string with a maximum length of 16 digits that maps to the OAdC field in the UCP `openSession` request.

Signature:

```
createOrUpdateCredentialMap(pluginInstanceId: String, appId: String, credentialId:
String)
```

Table 23–2 createOrUpdateCredentialMap

Parameter	Description
pluginInstanceId	ID of the plug-in instance being mapped; information about plug-in instances is maintained in the plug-in manager; see Operation:listPluginInstances for information about getting the list of plug-in instances IDs
appId	Application instance ID or Services Gatekeeper user to map to the remote host associated with the `pluginInstanceId`.
credentialId	ID of an existing credential entry in the configuration. If this value is empty or null, any existing mapping for the `appId` is removed.

Operation: createOrUpdateListenAddress

Scope: Cluster

Adds a listen address or updates an existing entry. If updating, the target list is updated to the new `targets` value passed by this call.

Signature:

```
createOrUpdateListenAddress(protocol: String, host: String, port: int, targets:
String)
```

Table 23–3 createOrUpdateListenAddress

Parameter	Description
protocol	Name of the protocol; for example, "ucp"
host	Host name or IP address; defaults to "localhost"
port	Port number to bind to
targets	Comma-separated list of server names or cluster names; defaults to all the targets in the cluster

Operation: createOrUpdateLocalHostAddress

Scope: Cluster

Adds a local host address or updates an existing entry.

A configured local host address is optional. When Services Gatekeeper connects to the remote network node, it uses the specified local host IP address/port combination to bind the socket on the Services Gatekeeper side of the connection. If the local host address is not configured, an ephemeral port is used.

When multiple connections are established with a network element, the port set by this operation is used as the starting offset.

When this operation is used to update an existing entry, the target list is changed to the new `targets` value passed by this call.

Signature:

```
createOrUpdateLocalHostAddress(pluginInstanceId: String, localHost: String,
localPort: int, targets: String)
```

Table 23–4 createOrUpdateLocalHostAddress

Parameter	Description
pluginInstanceId	ID of the plug-in instance being mapped. See Operation:listPluginInstances for information about getting the list of plug-in instance IDs.
localHost	Local host name or IP address
localPort	Local port number to bind to
targets	Comma-separated list of server names or cluster names; defaults to all the targets in the cluster

Operation: createOrUpdateRemoteHostAddress

Scope: Cluster

Creates or updates the connection details for the remote host to which the specified plug-in instance connects.

A remote host address is required to establish a connection.

Signature:

```
createOrUpdateRemoteHostAddress(pluginInstanceId: String, remoteHost: String,
remotePort: int)
```

Table 23–5 createOrUpdateRemoteHostAddress

Parameter	Description
pluginInstanceId	ID of the plug-in instance being mapped. See Operation:listPluginInstances for information about getting the list of plug-in instance IDs.
remoteHost	Remote host name or IP address
remotePort	Remote port number to bind to

Operation: createOrUpdateUserPasswordCredentialEntry

Scope: Cluster

Creates or updates the user and password parameters associated with the `credentialId`.

Signature:

```
createOrUpdateUserPasswordCredentialEntry(credentialId: String, remoteUser:
String, remotePassword: String)
```

Table 23–6 createOrUpdateUserPasswordCredentialEntry

Parameter	Description
credentialId	Unique identifier for the credential in the configuration
remoteUser	User name used to connect to the network node
remotePassword	Password used to connect to the network node

Operation: getAllListenAddress

Scope: Cluster

Returns the list of server addresses configured for this domain for the specified protocol.

Signature:

```
getAllListenAddress(protocol: String)
```

Table 23–7 getAllListenAddress

Parameter	Description
protocol	Protocol for which server addresses are configured in the current domain; for example, "ucp"

Operation: getConnectInfo

Scope: Server

Returns the ConnectInfo object of the remote host (SMSC or network node) for the server on which this MBean instance resides.

The ConnectInfo object includes:

- remoteHost
- remotePort
- localHost
- localPort
- credentialMap

The `localHost` and `localPort` values are server-specific.

The credentialMap maps application identifier with network node credentials; for example, with an SMSC credential.

Signature:

```
getConnectInfo(pluginInstanceId: String)
```

Table 23–8 getConnectInfo

Parameter	Description
pluginInstanceId	Plug-in instance ID for the remote host server for which connection information is requested. See Operation:listPluginInstances for information about getting the list of plug-in instance IDs.

Operation: getListenAddressForCurrentServer

Scope: Server

Returns a list of server addresses configured for the current server and the specified protocol.

Signature:

```
getListenAddressForCurrentServer(protocol: String)
```

Table 23–9 getListenAddressForCurrentServer

Parameter	Description
protocol	Protocol for which server addresses are configured in the current domain; for example, "ucp"

Operation: listAllCredentialEntries

Scope: Cluster

Returns the credential map containing the credential IDs and associated Credential objects in the configuration.

Signature:

```
listAllCredentialEntries()
```

Operation: removeConnectInfo

Scope: Cluster

Removes the remote host's connect address, local connect address, and credential mapping for the plug-in referenced by the `pluginInstanceId`.

Signature:

```
removeConnectInfo(pluginInstanceId: String)
```

Table 23–10 removeConnectInfo

Parameter	Description
pluginInstanceId	ID of the plug-in instance for which information is being removed. See Operation:listPluginInstances for information about getting the list of plug-in instance IDs.

Operation: removeCredentialEntry

Scope: Cluster

Removes the specified credential entry from the configuration.

Signature:

```
removeCredentialEntry(credentialId: String)
```

Table 23–11

Parameter	Description
credentialId	ID of an existing credential entry in the configuration

Operation: removeCredentialMap

Scope: Cluster

Removes the credential mapping between the `appId` and `pluginInstanceId` from the connect info configuration of the plug-in instance identified by the `pluginInstanceId`.

Signature:

```
removeCredentialMap(pluginInstanceId: String, appId: String)
```

Table 23–12 removeCredentialMap

Parameter	Description
pluginInstanceId	ID of the mapped plug-in instance that was mapped
appId	Application instance ID or Services Gatekeeper user mapped to the remote host associated with the `pluginInstanceId`

Operation: removeListenAddress

Scope: Cluster

Removes the specified listen address configuration from all the targets.

Signature:

```
removeListenAddress(protocol: String, host: String, port:int)
```

Table 23–13 removeListenAddress

Parameter	Description
protocol	Name of the protocol; for example, "ucp"
host	Host name or IP address
port	Port number to bind to

Operation: removeLocalHostAddress

Scope: Cluster

Removes the local host address information associated with the `pluginInstanceId`.

Signature:

```
removeLocalHostAddress(pluginInstanceId: String)
```

Table 23–14 removeLocalHostAddress

Parameter	Description
pluginInstanceId	ID of the plug-in instance for which information is being removed. See Operation:listPluginInstances for information about getting the list of plug-in instance IDs.

Operation: removeXParamFromCredentialEntry

Scope: Cluster

Removes the specified parameter from the specified credential entry.

Signature:

```
removeXParamFromCredentialEntry(credentialId: String, xParamName: String)
```

Table 23–15 removeXParamFromCredentialEntry

Parameter	Description
credentialId	Credential ID for the entry from which the parameter is removed.
xParamName	Name of XParam being removed.

24

Managing and Configuring Shortcode Mappings

This chapter describes how to configure shortcode mappings for incoming messages to Oracle Communications Services Gatekeeper.

Introduction

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.

Configuration and Management

All configuration and management is performed in the managed object, ShortCodeMapper.

The ShortCodeMapper OAM 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 > ShortCodeMapper
  - Plugin_px21_short_messaging_smpp > 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 need to 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

Management of short codes includes:

- Operation: addShortCodeMapping
- Operation: listShortCodeMappings
- Operation: removeShortCodeMapping

Reference: Attributes and Operations for ShortCodeMapper

Managed object: ShortCodeMapper

MBean: com.bea.wlcp.wlng.shortcode.management.ShortCodeMapperMBean

Following is a list of operations for configuration:

- Operation: addShortCodeMapping
- Operation: listShortCodeMappings
- Operation: removeShortCodeMapping

Operation: addShortCodeMapping

Scope: Cluster

Translates the original destination address of a network-triggered message to a translated destination address. The translated destination address is used when the message is matched with notification registrations. This makes it possible for an application to register for a notification from a single shortcode and receive messages sent to a range of short codes (original destination addresses).

> **Note:** Shortcode translation takes place before mapping a mobile originated message to registered application notifications. If two or more short code translations match an original destination number the best match, the most specific, is chosen.

An identifier for the shortcode translation is returned.

Signature:

```
addShortCodeMapping(orig_dest_pattern: String, trans_dest_addr: String)
```

Table 24–1 addShortCodeMapping

Parameter	Description
orig_dest_pattern	Pattern that is matched against the original destination address of a message received from the network. The pattern should be specified as a regular expression: ^6
trans_dest_addr	The resulting translated address. Addresses must be specified with the prefix “tel:”.

Operation: listShortCodeMappings

Scope: Cluster

Displays a list of shortcode mappings.

Signature:

```
listShortCodeMappings()
```

Operation: removeShortCodeMapping

Scope: Cluster

Removes a previously added shortcode mapping. Both fields must match.

Signature:

```
removeShortCodeMapping(OriginalDestination: String, MappedDestination: String)
```

Table 24–2 removeShortCodeMapping

Parameter	Description
OriginalDestination	Registered original destination pattern.
MappedDestination	Registered translated destination address.

25

Managing OSA/Parlay Gateway Connections using Parlay_Access

This chapter describes Open Services Architecture (OSA)/Parlay Gateways and explains how to connect them to OCSG.

Understanding OSA/Parlay Gateway and account mappings

This section describes the general model OCSG 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 towards 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**, identified by a `gatewayId`, which represents the actual OSA/Parlay Gateway. Each OSA Gateway that is 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 system-wide 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**, which means that the connection between Services Gatekeeper and a specific SCS in the OSA/Parlay Gateway is active and functional.
- **Inactive**, which means that 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 in conjunction 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 towards 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 non-production environments, the WebLogic Server CertGen utility can be used to create certificates and keys.

Overall workflow when connecting to an OSA Gateway

Follow the steps below 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".
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 information specified in "Operation: addGw"

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 Operation: addConnection

   If the IOR is provided as a String: use Operation: addConnectionIOR

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 the information specified in Operation: addClient

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 the information specified in Operation: addMapping
3. Click **Invoke**.

   The OSA client mapping is created.

Reference: Attributes and Operations for Parlay_Access

Managed object: Container Services->Parlay_Access_*Communication Service*

Where Communication Service is one of:

- audio_call_px30
- call_notification_px30
- third_party_call_px30

> **Note:** There are three MBeans, one for each communication service of Parlay type. It does not matter which one you use. They all operate on the same data.

MBean: com.bea.wlcp.wlng.parlay.access.ParlayAccessMBean

Following is a list of attributes and operations for configuration and maintenance:

- Attribute: EricssonAuthentication
- Operation: activateMapping
- Operation: addClient
- Operation: addConnection
- Operation: addConnectionIOR
- Operation: addGw
- Operation: addMapping
- Operation: listActiveMappings
- Operation: listActiveMappingsForGw
- Operation: listGw
- Operation: listMappings
- Operation: removeClient
- Operation: removeConnection
- Operation: removeGw
- Operation: removeMapping

- Operation: setKeyStorePassword
- Operation: viewActiveMappingState

Attribute: EricssonAuthentication

Scope: Cluster

Unit: Not applicable

Format: Boolean

Set to:

- `true` if connecting to an Ericsson OSA/Parlay Gateway
- `false` if connecting to other gateways.

Operation: activateMapping

Scope: Cluster

Activates an existing mapping.

Signature:

```
activateMapping(id: String)
```

Table 25–1 activateMapping

Parameter	Description
id	ID of the OSA/Parlay client mapping to activate. See "Operation: listMappings".

Operation: addClient

Scope: Cluster

Adds an OSA/Parlay Client.

Signature:

```
addClient(osaClientAppId: String, clientKeyFile: String, clientCertFile: String,
clientKeyPwd: String, keystorePwd: String)
```

Table 25–2 addClient

Parameter	Description
osaClientAppId	The Enterprise Operator ID and Application ID registered for the OSA/Parlay Client in the OSA/Parlay Gateway. This value must be unique. The format is: `<Enterprise Operator>\<Application ID>` Example: myEntopId\myAppId
clientKeyFile	The directory path (including file name) to the private key for the OSA Client. **Note:** This path is on the file system of the Services Gatekeeper Network Tier server. Leave empty if not authenticating using certificates.

Table 25–2 (Cont.) addClient

Parameter	Description
clientCertFile	The directory path (including file name) to the certificate for the OSA Client. The certificate is provided to verify the private key is correct. **Note:** This path is on the file system of the Services Gatekeeper Network Tier server. Leave empty if not authenticating using certificates.
clientKeyPwd	The password for the private key. Leave empty if not authenticating using certificates.
keystorePwd	The keystore's password as defined when configuring the Services Gatekeeper: see "Operation: setKeyStorePassword".

Operation: addConnection

Scope: Cluster

Adds a connection to a Framework in the OSA/Parlay Gateway using a file that contains the name service IOR.

Signature:

```
addConnection(gwId: int, nsRef: String, nsName: String, initialRef: String,
priority: int)
```

Table 25–3 addConnection

Parameter	Description
gwId	The ID of the OSA/Parlay Gateway, as returned when the OSA Gateway was created. See "Operation: addGw". Also see "Operation: listGw".
nsRef	The directory path (including file name) for the file containing the name service IOR. Leave blank if `initialRef` is specified.
nsName	The name of the initial object in the name service. For example: `parlay_initial`. Use path syntax to specify recursive naming contexts. For example: `/parlay/fw/parlay_inital` Leave blank if `initialRef` is specified.
initialRef	The directory path, including file name, for the file containing the IOR to the Parlay initial object. Leave blank if `nsRef` and `nsName` is specified.
priority	Priority of this connection. Should be unique across all connections. The lower the number, the higher the priority.

Operation: addConnectionIOR

Scope: Cluster

Adds a connection to a Framework in the OSA/Parlay Gateway using an IOR string.

Signature:

```
addConnectionIOR(gwId: int, ior: String, ns: String, priority: int)
```

Table 25–4 addConnectionIOR

Parameter	Description
gwId	The ID of the OSA/Parlay Gateway, as returned when the OSA Gateway was created. See "Operation: addGw". Also see "Operation: listGw".
ior	IOR string of either the NS or the initial object.
ns	The name of the initial object in the name service. For example: `parlay_initial` Use path syntax to specify recursive naming contexts. For example: `/parlay/fw/parlay_inital` Leave blank if IOR to the initial object is specified.
priority	Priority of this connection. Should be unique across all connections. The lower the number, the higher the priority.

Operation: addGw

Scope: Cluster

Adds an OSA/Parlay Gateway to be used by the OSA/Parlay type plug-ins. More than one Gateway can be added.

Signature:

```
addGw(name: String, osaFwCert: String, reAuthWaitTime: int, keystorePwd: String)
```

Returns the ID for the OSA Gateway. This ID is used to create an OSA/Parlay Gateway Connection: see "Operation: addConnection" and to create an OSA/Parlay client mapping: see "Operation: addMapping".

Table 25–5 addGw

Parameters	Description
name	Descriptive name of the OSA Gateway.
osaFwCert	The certificate to use when connecting to the OSA Gateway's Framework. The certificate is supplied by the OSA Gateway administrator. Leave empty if not authenticating using certificates.
reAuthWaitTime	The time to wait before reattempting to authenticate and obtain OSA Service Managers if all connections to the OSA Gateway are lost. Given in seconds
keystorePwd	The password for the Services Gatekeeper keystore.

Operation: addMapping

Scope: Cluster

Adds an OSA client mapping.

Signature:

```
addMapping(serviceProviderID: String, applicationID: String, serviceType: String,
osaClientAppId: String, properties: String, authType: String, encryptionMethod:
String, signingAlgorithm: String, gatewayId: int, initConnection: boolean)
```

Table 25–6 addMapping

Parameter	Description
serviceProviderID	ID of the service provider account the application is associated with. **Note:** If left empty, the mapping will *not* be applied on the service provider account and application account levels.
applicationID	ID of the application account. **Note:** If left empty, the mapping will *not* be applied on the application account level.
serviceType	OSA/Parlay service type name (`TpServiceTypeName`) of the OSA/Parlay SCS to which the OSA Client is to be mapped. See the specification for the OSA/Parlay Framework for a list of recommended service type names.
osaClientAppId	The OSA/Parlay account's `clientAppID`, a string consisting of the entOpId followed by `\`, followed by the appId. For example: `sp1\app1`. The entOpId and appId are provided by the OSA Gateway administrator.
properties	OSA/Parlay service properties to be used in the lookup (service discovery) phase when requesting a service (OSA/Parlay SCS) from the OSA/Parlay Gateway. The properties are specified as a space-separated list in the following way: *propname1 propval1 propname2 propval2* The properties vary between OSA/Parlay Gateway implementations.
authType	Authentication type to be used. The type is defined according to the OSA/Parlay standard. P_AUTHENTICATION is the only supported value. **Note:** When `P_AUTHENTICATION` is used, no encryption or signing algorithm will be used and the `encryptionMethod` and `signingAlgorithm` parameters can be left empty.
encryptionMethod	Method used for encryption. The type is defined according to OSA/Parlay standard. If the type is not specified, enter `P_RSA_1024`.
signingAlgorithm	Signing algorithm. The type is defined according to OSA/Parlay standard. If the type is not specified, enter `P_MD5_RSA_1024`.
gatewayId	OSA/Parlay Gateway ID. This ID was generated when the OSA/Parlay Gateway was created: see "Operation: addGw" and "Operation: listGw".
initConnection	Indicating if the connection to OSA/Parlay Gateway should be initialized immediately. That is, if authentication should performed when the "Operation: addClient" operation is invoked.

Operation: listActiveMappings

Scope: Cluster

Lists the IDs for active OSA/Parlay client mappings.

Signature:

```
listActiveMappings()
```

Returns a list of IDs for active mappings.

Table 25–7 listActiveMappings

Parameter	Description
Not applicable	Not applicable

Operation: listActiveMappingsForGw

Scope: Cluster

Lists the IDs of all active OSA/Parlay client mappings for a specific OSA/Parlay Gateway.

Signature:

```
listActiveMappingsForGw(gwId: int)
Returns a list of IDs for active mappings for the Gateway.
```

Table 25–8 listActiveMappingsForGw

Parameter	Description
gwId	The ID of the OSA Gateway.

Operation: listGw

Scope: Cluster

Lists the IDs of all registered OSA/Parlay Gateways.

Signature:

```
listGw()
```

Operation: listMappings

Scope: Cluster

Lists the configured OSA/Parlay client mappings.

Signature:

```
listMappings()
```

Operation: removeClient

Scope: Cluster

Removes an OSA/Parlay client.

Signature:

```
removeClient(osaClientAppId: String, keystorePwd: String)
```

Table 25–9 removeClient

Parameter	Description
osaClientAppId	The OSA/Parlay client application ID (and alias in keystore). See "Operation: addClient".
keystorePwd	The Services Gatekeeper keystore password.

Operation: removeConnection

Scope: Cluster

Removes an OSA/Parlay Gateway connection.

Signature:

```
removeConnection(gatewayId: int, connectionId: int)
```

Table 25–10 removeConnection

Parameter	Description
gatewayId	The ID of the OSA/Parlay Gateway.
connectionId	The ID of the connection. The ID was returned when the connection was set up. See: ■ Operation: addConnection ■ Operation: listActiveMappings ■ Operation: listMappings

Operation: removeGw

Scope: Cluster

Removes an OSA/Parlay Gateway.

Signature:

```
removeGw(id: int, keystorePwd: String)
```

Table 25–11 removeGw

Parameter	Description
id	The ID of the OSA/Parlay Gateway to remove. The ID returned when the OSA Gateway was created, see: ■ Operation: addGw ■ Operation: listGw
`keystorePwd`	The Services Gatekeeper keystore password.

Operation: removeMapping

Scope: Cluster

Removes an OSA/Parlay client mapping.

Signature:

```
removeMapping(id: int)
```

Table 25–12 removeMapping

Parameter	Description
id	ID of the OSA/Parlay client mapping to remove.

Operation: setKeyStorePassword

Scope: Cluster

Sets the password that protects the keystore.

Signature:

```
setKeyStorePassword(newPassword: String, oldPassword: String)
```

Table 25–13 setKeyStorePassword

Parameter	Description
newPassword	The new password for the keystore.
oldPassword	The old password for the keystore.

Operation: viewActiveMappingState

Scope: Cluster

Displays the state of an active mapping OSA/Parlay client mapping.

Signature:

```
viewActiveMappingState(mappingId: int)
```

Table 25–14 viewActiveMappingState

Parameter	Description
mappingId	The ID of the OSA client mapping.

26

Resolving Policy Deny Codes

This chapter explains the policy deny codes generated by Service Interceptors.

Introduction

Policy deny codes are embedded in Policy Exception 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 towards 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>. Example: <denyCode>21</denyCode>.

Policy Deny Data

Table 26–1 contains a list of policy deny codes data generated when a policy deny occurs.

Table 26–1 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.
6	102827	1002	Unable to get service provider and application information.

Table 26–1 (Cont.) Policy deny data

Policy Deny Code	ID of corresponding alarm	TagPolicy field in alarm	Description
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 is not active.
12	102826	1001	Application instance is 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 is empty.
23	102833	3002	Accessing the method is 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.
31	102838	3005	Application or Service provider group service contract is out of date.

Table 26–1 (Cont.) Policy deny data

Policy Deny Code	ID of corresponding alarm	TagPolicy field in alarm	Description
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	Subscriber restrict all.
36	102842	2007	Subscriber quota limit reached.
37	102842	2007	Subscriber 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.

27

Filtering Tunneled Parameters

This chapter describes an application that filters tunneled parameters. This enhancement enables tighter security by allowing only xparameters that are explicitly allowed.

For general information about xparameters, see the discussion of parameter tunneling in the *Oracle Communications Services Gatekeeper System Application Developer's Guide*.

About the XParameter Filter Application

The filter application blocks requests that contain xparameters that are not configured as allowed xparameters. Filtering is on a global, not application, level.

The application is implemented by the **x-param-filter-interceptor** interceptor. When the application is on and 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.

The application is deployed as a standalone EAR file: **interceptor_xparam/xparam_interceptors.ear**.

The filter application, named **interceptor_xparam**, is installed with Services Gatekeeper. To turn the application on, configure the customized interceptor chain and enable **interceptor_xparam**. When the filter application is on, you need to configure the xparameters that you want to allow as described in "XParameter Filter Configuration File"; otherwise all requests that contain xparameters will be rejected.

XParameter Filter Configuration File

The xparameter filter configuration file is **xparam_filter_config.xml**. It is located in **interceptor_xparam/xparam_interceptors.ear./APP-INF/classes**.

To allow an xparameter in a request, list its key as an `<xParamKey>` sub element in 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 *Oracle Communications Services Gatekeeper Communication Service 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 it will be rejected.

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

```
<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, make sure that any custom interceptor that adds xparameters is executed after the **x-param-filter-interceptor** in the interceptor stack

See the discussion of the service interceptors in the *Oracle Communications Services Gatekeeper Platform Development Studio Developer's Guide* for information about the order of execution of custom interceptors.

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